The Case for Common Flaw Enumeration

Robert A. Martin
MITRE Corporation
202 Burlington Road
Bedford, MA 01730
1-781-271-3001
ramartin@mitre.org

Steven M. Christey
MITRE Corporation
202 Burlington Road
Bedford, MA 01730
1-781-271-3961
coley@mitre.org

Joe Jarzombek
National Cyber Security Division
Department of Homeland Security
Arlington, VA 22201
1-703-235-5126
joe.jarzombek@dhs.gov

ABSTRACT

Software acquirers want assurance that the software products they are obtaining are reviewed for known types of security flaws. The acquisition groups in large government and private organizations are moving forward to use these types of reviews as part of future contracts. The tools and services that can be used for this type of review are fairly new at best. However, there are no nomenclature, taxonomies, or standards to define the capabilities and coverage of these tools and services. This makes it difficult to comparatively decide which tool/service is best suited for a particular job. A standard taxonomy of software security vulnerabilities can serve as a unifying language of discourse and measuring stick for tools and services. Leveraging the diverse thinking on this topic from academia, the commercial sector, and government, we can pull together the most valuable breadth and depth of content and structure to serve as a unified standard. As a starting point, we plan to leverage the wide acceptance and use of the Common Vulnerabilities and Exposures (CVE) list of publicly known software security flaws. In conjunction with industry and academia, we propose to extend the coverage of the CVE concept [1] into security-based code assessment tools and services. Our objective is to help shape and mature this new code security assessment industry and also dramatically accelerate the use and utility of these capabilities for organizations in reviewing the software systems they acquire or develop.

Categories and Subject Descriptors

D.2.4 [Software Engineering]: Software/Program Verification

General Terms

Software, Security, Testing, Verification, Flaws, Faults.

Keywords

taxonomies, static analysis, security flaws, weaknesses, idiosyncrasies, WIFF, Common Vulnerabilities and Exposures, CVE, vulnerabilities, secure software, software security assurance.

1. INTRODUCTION

More and more organizations want assurance that the software products they acquire and develop are free of known types of security flaws. High quality tools and services for finding security flaws in code are new. The question of which tool/service is appropriate/better for a particular job is hard to answer given the lack of structure and definition in the code assessment industry.

There are several efforts currently ongoing to begin to resolve some of these shortcomings including the Department of Homeland Security (DHS) National Cyber Security Division (NCSD) sponsored Software Assurance Metrics and Tool Evaluation (SAMATE) project [2] being led by the National Institute of Standards and Technology (NIST), and the Department of Defense (DOD) sponsored Code Assessment Methodology Project (CAMP) which is part of the Protection of Vital Data (POVD) effort [3] being conducted by Concurrent Technologies Corporation (CTC), among others. While these efforts are well placed, timely in their objectives and will surely yield high value in the end, they both would benefit from a common description of the underlying security vulnerabilities in software that they are targeted to resolve. Without such a common taxonometric description, many of these efforts cannot move forward in a meaningful fashion or be aligned and integrated with each other to provide strategic value.

Past efforts at developing this kind of taxonomy have been limited by a very narrow technical domain focus or have largely focused on high-level theories, taxonomies, or schemes that do not reach the level of detail or variety of security issues that are found in today's products. As an alternate approach, under sponsorship of DHS NCSD, MITRE investigated the possibility of leveraging the CVE initiative's experience in analyzing nearly 13,000 real-world vulnerabilities reported and discussed by industry and academia.

As part of the creation of the CVE List, over the last five years MITRE's CVE initiative, sponsored by DHS NCSD, has developed a preliminary classification and categorization of vulnerabilities, attacks, faults, and other concepts that can be used to help define this arena. However, the current groupings used in the development of CVE, while sufficient for that task, are too rough to be used to identify and categorize the functionality offered within the offerings of the code security assessment industry. Additional fidelity and succinctness is needed to support this type of usage and there needs to be additional details and description for each of the different nodes and groupings such as the effects, behaviors, and implementation details, etc.

As part of MITRE's participation in the DHS-sponsored NIST SAMATE project MITRE took a first cut at revising the internal CVE category work for usage in the code assessment industry. The resultant document, called the Preliminary List Of Vulnerability Examples for Researchers (PLOVER) [4], is a working document that lists over 1,400 diverse, real-world examples of vulnerabilities, identified by their CVE name. The vulnerabilities are organized within a detailed conceptual framework that currently enumerates 290 individual types of Weaknesses, Idiosyncrasies, Faults, Flaws (WIFFs), with a large number of real-world vulnerability examples for each type of WIFF. PLOVER represents the first cut of a truly bottom-up effort to take real-world observed faults and flaws that *do* exist in code, abstract them and group them into common classes representing more general potential vulnerabilities that *could* exist in code, and then finally to organize them in an appropriate relative structure so as to make them accessible and useful to a diverse set of audiences for a diverse set of purposes. The initial details of this enumeration can be found at the end of this paper.

Working with the community under the NIST SAMATE project, we are establishing acceptable definitions and descriptions of these CWEs. When completed, this will serve as a mechanism for describing code vulnerability assessment capabilities in terms of their coverage of the different CWEs. If necessary, this will also be scoped to specific languages, frameworks, platforms and machine architectures. More work is required to group PLOVER WIFFs into a taxonomy more useful for SAMATE.

2. OBJECTIVES

As discussed above, we are leveraging PLOVER as a starting point for the creation of a formal enumeration of the set of software security Weaknesses, Idiosyncrasies, Faults, Flaws (WIFFs) to serve as a common language for describing software security vulnerabilities, to serve as a standard measuring stick for software security tools targeting these vulnerabilities, and to provide a common glue for vulnerability identification, mitigation and prevention efforts. When complete, this Common WIFF Enumeration (CWE) will not only encompass a large portion of the CVE List's 12,000 plus CVE names but it will also include detail and breadth from a diverse set of other industry and academic sources and examples. Once a comprehensively broad set of CWEs has been identified and collected, we will again look to these other sources and examples for approaches to organizing this enumeration in order to provide more simplicity to various potential users through taxonometric layering.

Working with the community under the DHS-sponsored NIST SAMATE project we are proceeding to establish acceptable definitions and descriptions of these CWEs to support finding these types of software security flaws in code prior to fielding. When completed this will be a mechanism for describing each of the industry's software security flaw code assessment capabilities in terms of their coverage of the different CWEs. If necessary, this will also be scoped to specific languages, frameworks, platforms and machine architectures.

Additionally, we are working with researchers and software suppliers to determine what sort of metadata and resources (e.g. code exemplars, patterns, code snippets, etc.) will be needed to allow tools to be tailored or enhanced to identify CWEs in code. This work will also align with and leverage the SAMATE project's various sub-efforts including its development of a corpus of data to determine precision and recall statistics for verifying the effectiveness of these types of code assessment tools with respect to finding CWEs.

Beyond the creation of the vulnerability taxonomy for the stated reasons, a further end goal of this effort will be to take the findings and results of this work and roll them into the CVE initiative as the foundation of a new type of compatibility that can be directly used by organizations in their selection and evaluation of tools and/or services for assessing their acquired software for known types of flaws.

3. APPROACH

A main theme of this effort is to leverage the existing work on this topic area [5]-[14] in light of the large number of diverse real-world vulnerabilities in CVE. We will leverage as many sources and examples as we can gain access to as well as collaborate with key industry players who are currently tackling this subject. We will work in conjunction with researchers at the NIST, The Open Web Application Security Project (OWASP), Ounce Labs, Cigital, Fortify Software, Cenzic, Microsoft, Klocwork, and Secure Software, and other interested parties, to develop specific and succinct definitions of the CWE list elements that adequately describe and differentiate the various CWEs while capturing their specific effects, behaviors, exploit mechanisms, and implementation details. In addition, we will assign the appropriate CWE to the CVE names so that each CWE group will have a list of the CVE names that belong to that CWE category of software security flaws. In constructing the CWE list, we will strive for maximum comprehensive coverage across appropriate conceptual, business and technical domains.

In our efforts to define organizational structure to the CWE list elements, we will look not only to PLOVER, but also to leading thoughts in this area including the McGraw/Fortify "Kingdoms" taxonomy [15], Howard, LeBlanc & Viega's *19 Deadly Sins* [16], Secure Software's CLASP [17], among others. In defining the organizational structure, we will strive for simplicity and appropriateness of description for leveraging by various audiences and for various purposes through the use of taxonometric layering. We currently foresee using a three tiered approach, in which the lowest level consists of the full CWE list (likely hundreds of nodes) and that is applicable to tool vendors and detailed research efforts. The middle tier would consist of descriptive affinity groupings of CWEs (likely 25-50 nodes) that are useful to software security and software development practitioners. The top level would consist of high-level groupings of the middle tier nodes (likely 5-10 nodes) to define strategic classes of vulnerability and is useful for high level discourse among software practitioners, business people, tool vendors, researchers, etc.

Once an initial CWE list and organizational structure have been defined, we will collaborate with our colleagues in the industry to further refine the required attributes of CWE list elements into a more formal schema defining the metadata structure necessary to support the various uses of the taxonomy. This schema will also be driven by a desire to align with and support the other SAMATE and CAMP efforts such as software metrics, software security tool metrics, the software security tool survey, the methodology for validating software security tool claims, and the reference datasets.

With a schema defined, an initial comprehensive list of CWEs identified and defined and an organizational structure in place, this set of content will be submitted to a much broader audience of industry participants to discuss, review and revise. This cycle will iterate until a general consensus can be reached on what will become the first release of the specification (a defacto standard).

4. IMPACT AND TRANSITION OPPORTUNITIES

The completion of this effort will yield consequences of three types: direct impact and value, alignment with and support of other existing efforts, and enabling of new follow-on efforts to provide value that is not currently being pursued.

Following is a list of the direct impacts this effort will yield. Each impact could be the topic of much deeper ongoing discussion.

1. Provide a common language of discourse for discussing, finding and dealing with the causes of software security vulnerabilities as they are manifested in code.
2. Allow software security tool vendors and service providers to make clear and consistent claims of the security vulnerability causes that they cover to their potential user communities in terms of the CWEs that they look for in a particular code language. Additionally, a new type of CVE Compatibility will be developed to allow security tool and service providers to publicly declare their capability's coverage of CWEs
3. Allow purchasers to compare, evaluate and select software security tools and services that are most appropriate to their needs including having some level of assurance of the level of CWEs that a given tool would find. Software purchasers would be able to compare coverage of tool and service offerings against the list of CWEs and the programming languages that are used in the software they are acquiring.
4. Enable the verification of coverage claims made by software security tool vendors and service providers (this is supported through CWE metadata and alignment with the SAMATE reference dataset).
5. Enable government and industry to leverage this standardization in the contractual terms and conditions.

Following is a list of alignment opportunities with existing efforts that are provided by the results of this effort. Again, each of these items could be the topic of much deeper ongoing discussion.

1. Mapping of CWEs to CVEs. This mapping will help bridge the gap between the potential sources of vulnerabilities and examples of their observed instances providing concrete information for better understanding the CWEs and providing some validation of the CWEs themselves.
2. Bidirectional alignment between the vulnerability taxonomy and the SAMATE metrics effort.
3. The SAMATE software security tool/service capability framework effort that is tasked with designing a framework and schema to quantitatively and qualitatively describe the capabilities of tools and services would be able to leverage this vulnerability taxonomy as the core layer of the framework. This framework effort is not an explicitly called out item in the SAMATE charter but is implied as necessary to meet the project's other objectives.
4. The SAMATE software security tool and services survey effort would be able to leverage this vulnerability taxonomy as part of the capability framework to effectively and unambiguously describe various tools and services in a consistent apples-to-apples fashion.
5. There should be bidirectional alignment between this source of vulnerability taxonomy and the SAMATE reference dataset effort such that CWEs could reference supporting reference dataset entries as code examples of that particular CWE for explanatory purposes and reference dataset entries could reference the associated CWEs that they are intended to demonstrate for validation purposes. Further, by working with industry, an appropriate method could be developed for collecting, abstracting, and sharing code samples from the code of the products that the CVE names are assigned to with the goal of gathering these code samples from industry researchers and academia so that they could be shared as part of the reference dataset and aligned with the vulnerability taxonomy. These samples would then be available as tailoring and enhancement aides to the developers of code assessment security tools. We could actively engage closed source and open source development organizations that work with the CVE initiative to assign CVE names to vulnerabilities to identify an approach that would protect the source of the samples while still allowing us to share them with others. By using the CVE-based relationships with these organizations, we should be able to create a high-quality collection of samples while also improving the accuracy of the security code assessment tools that are available to the software development groups to use in vetting their own product's code
6. The SAMATE software security tool/service assessment framework effort that is tasked with designing a test and validation framework to support the validation of tool/service vendor claims by either the purchaser directly or through a 3rd party, would rely heavily on this sources of vulnerability taxonomy as its basis of analysis. To support this, we would work with researchers to define the mechanisms used to exploit the various CWEs for the purposes of helping to clarify the CWE groupings and as a possible verification method for validating the effectiveness of the tools that identify the presence of CWEs in code by exploring the use of several testing approaches on the executable version of the reviewed code. The effectiveness of these test approaches could be explored with the goal of identifying a method or methods that are effective and economical to apply to the validation process
7. Bidirectional mapping between CWEs and Coding Rules, such as those deployed as part of the DHS NCSD "Build Security In" (BSI) website [18], used by tools and in manual code inspections to identify vulnerabilities in software.
8. There should be bidirectional alignment between the vulnerability taxonomy and the CAMP malware repository effort similar to the alignment with the SAMATE reference dataset described in #5 above.

Following is a list of new, unpursued follow-on opportunities for creating added value to the software security industry.

1. Expansion of the Coding Rules Catalog on the DHS BSI website to include full mapping against the CWEs for all relevant technical domains.

2. Identification and definition of specific domains (language, platform, functionality, etc.) and relevant protection profiles based on coverage of CWEs. These domains and profiles could provide a valuable tool to security testing strategy and planning efforts.

With this fairly quick research and refinement effort, this work should be able to help shape and mature this new code security assessment industry, and dramatically accelerate the use and utility of these capabilities for organizations and the software systems they acquire, develop, and use.

5. Initial Weaknesses, Idiosyncrasies, Faults, Flaws (WIFFs) Enumeration

The following section introduces the current content we have derived through studying a large portion of the CVE list. The listing below, which is comprised of 290 specific types of weakness, idiosyncrasies, faults and flaws (WIFFs) is not exhaustive and will certainly evolve.

Our purpose in coining the term "WIFFs" is avoid the use of the term "vulnerability" for these items. The term "vulnerability" is frequently used in the community to apply to other concepts including bugs, attacks, threats, risks, and impact. Also, there are widely varying opinions regarding what "risk level" must be associated with a problem in order to call it a vulnerability, e.g. in terms of denial-of-service attacks and minor information leaks. Finally, not every instance of the items listed below, or those collected in this overall effort, will need to be removed or addressed in the applications they reside in. While they most certainly need to be examined and evaluated for their potential impact to the application, there will certainly be a large number of these items that could be safely left as is, or dealt with by making some minimal adjustments or compensations to keep them from manifesting into exploitable vulnerabilities. If we went forward using the term "vulnerability" for these items, there would be a built-in bias and predisposition to remove and eliminate each and every one of them, which would be a massive and unnecessary waste of time and resources.

The items below have not been categorized except in the most obvious and expeditious manner. With the incorporation of the other contributions from academia and industry sources we will most certainly reorganize these groupings as more examples and specifics are added. With this caveat we provide the following summary of the 28 main categories which contain the 290 individual types of WIFFs we have enumerated to-date.

1. Buffer overflows, format strings, etc. [BUFF] (10 types)

   These categories cover the increasingly diverse set of WIFFs that are generally referred to as "buffer overflows." The specific types in this group are: Buffer Boundary Violations ("buffer overflow"), Unbounded Transfer ("classic overflow"), Boundary beginning violation ("buffer underflow"), Out-of-bounds Read, Buffer over-read, Buffer under-read, Array index overflow, Length Parameter Inconsistency, Other length calculation error, Format string vulnerability

2. Structure and Validity Problems [SVM] (10 types)

   These categories cover certain ways in which "well-formed" data could be malformed. The specific types in this group are: Missing Value Error, Missing Parameter Error, Missing Element Error, Extra Value Error, Extra Parameter Error, Undefined Parameter Error, Undefined Value Error, Wrong Data Type, Incomplete Element, Inconsistent Elements

3. Special Elements (Characters or Reserved Words) [SPEC] (19 types)

   These categories cover the types of special elements (special characters or reserved words) that become security-relevant when transferring data between components. The specific types in this group are: General Special Element Problems, Parameter Delimiter, Value Delimiter, Record Delimiter, Line Delimiter, Section Delimiter, Input Terminator, Input Leader, Quoting Element, Escape, Meta, or Control Character / Sequence, Comment Element, Macro Symbol, Substitution Character, Variable Name Delimiter, Wildcard or Matching Element, Whitespace, Grouping Element / Paired Delimiter, Delimiter between Expressions or Commands, Null Character / Null Byte

4. Common Special Element Manipulations [SPECM] (11 types)

   These categories include different ways in which special elements could be introduced into input to software as it operates. The specific types in this group are: Special Element Injection, Equivalent Special Element Injection, Leading Special Element, Multiple Leading Special Elements, Trailing Special Element, Multiple Trailing Special Elements, Internal Special Element, Multiple Internal Special Element, Missing Special Element, Extra Special Element, Inconsistent Special Elements

5. Technology-Specific Special Elements [SPECTS] (17 types)

   These categories cover special elements in commonly used technologies and their associated formats. The specific types in this group are: Cross-site scripting (XSS), Basic XSS, XSS in error pages, Script in IMG tags, XSS using Script in Attributes, XSS using Script Via Encoded URI Schemes, Doubled character XSS manipulations, e.g. "<<script", Null Characters in Tags, Alternate XSS syntax, OS Command Injection, Argument Injection or Modification, SQL injection, LDAP injection, XML injection (aka Blind Xpath injection), Custom Special Character Injection, CRLF Injection, Improper Null Character Termination

6. Pathname Traversal and Equivalence Errors [PATH] (47 types)

   These categories cover the use of file and directory names to either "escape" out of an intended restricted directory, or access restricted resources by using equivalent names. The specific types in this group are: Path Traversal, Relative Path Traversal, "/directory/../filename", "../filedir",

"/../filedir", "directory/../../filename", "..\filename" ("dot dot backslash"), "\..\filename" ("leading dot dot backslash"), "\directory\..\filename", "directory\..\..\filename", "..." (triple dot), "...." (multiple dot), "...//" (doubled dot dot slash), Absolute Path Traversal, /absolute/pathname/here, ".../...//", \absolute\pathname\here ("backslash absolute path"), "C:dirname" or C: (Windows volume or "drive letter"), "\\UNC\share\name\" (Windows UNC share), Path Equivalence, Trailing Dot - "filedir.", Internal Dot - "file.ordir", Multiple Internal Dot - "file...dir", Multiple Trailing Dot - "filedir....", Trailing Space - "filedir ", Leading Space - " filedir", file[SPACE]name (internal space), filedir/ (trailing slash, trailing /), //multiple/leading/slash ("multiple leading slash"), /multiple//internal/slash ("multiple internal slash"), /multiple/trailing/slash// ("multiple trailing slash"), \multiple\\internal\backslash, filedir\ (trailing backslash), /./ (single dot directory), filedir* (asterisk / wildcard), dirname/fakechild/../realchild/filename, Windows 8.3 Filename, Link Following, UNIX symbolic link (symlink) following, UNIX hard link, Windows Shortcut Following (.LNK), Windows hard link, Virtual Files, Windows MS-DOS device names, Windows ::DATA alternate data stream, Apple ".DS_Store", Apple HFS+ alternate data stream

7. Channel and Path Errors [CP] (13 types)

   These categories cover the ways in which the use of communication channels or execution paths could be security-relevant. The specific types in this group are: Channel Errors, Unprotected Primary Channel, Unprotected Alternate Channel, Alternate Channel Race Condition, Proxied Trusted Channel, Unprotected Windows Messaging Channel ("Shatter"), Alternate Path Errors, Direct Request aka "Forced Browsing", Miscellaneous alternate path errors, Untrusted Search Path, Mutable Search Path, Uncontrolled Search Path Element, Unquoted Search Path or Element

8. Cleansing, Canonicalization, and Comparison Errors [CCC] (16 types)

   These categories cover various ways in which inputs are not properly cleansed or canonicalized, leading to improper actions on those inputs. The specific types in this group are: Encoding Error, Alternate Encoding, Double Encoding, Mixed Encoding, Unicode Encoding, URL Encoding (Hex Encoding), Case Sensitivity (lowercase, uppercase, mixed case), Early Validation Errors, Validate-Before-Canonicalize, Validate-Before-Filter, Collapse of Data into Unsafe Value, Permissive Whitelist, Incomplete Blacklist, Regular Expression Error, Overly Restrictive Regular Expression, Partial Comparison

9. Information Management Errors [INFO] (19 types)

   These categories involve the inadvertent or intentional publication or omission of sensitive data, which is not resultant from other types of WIFFs. The specific types in this group are: Information Leak (information disclosure), Discrepancy Information Leaks, Response discrepancy infoleak, Behavioral Discrepancy Infoleak, Internal behavioral inconsistency infoleak, External behavioral inconsistency infoleak, Timing discrepancy infoleak, Product-Generated Error Message Infoleak, Product-External Error Message Infoleak, Cross-Boundary Cleansing Infoleak, Intended information leak, Process information infoleak to other processes, Infoleak Using Debug Information, Sensitive Information Uncleared Before Use, Sensitive memory uncleared by compiler optimization, Information loss or omission, Truncation of Security-relevant Information, Omission of Security-relevant Information, Obscured Security-relevant Information by Alternate Name

10. Race Conditions [RACE] (6 types)

    These categories cover various types of race conditions. The specific types in this group are: Race condition enabling link following, Signal handler race condition, Time-of-check Time-of-use race condition, Context Switching Race Condition, Alternate Channel Race Condition, Other race conditions

11. Permissions, Privileges, and ACLs [PPA] (20 types)

    These categories include the improper use, assignment, or management of permissions, privileges, and access control lists. The specific types in this group are: Privilege / sandbox errors, Incorrect Privilege Assignment, Unsafe Privilege, Privilege Chaining, Privilege Management Error, Privilege Context Switching Error, Privilege Dropping / Lowering Errors, Insufficient privileges, Misc. privilege issues, Permission errors, Insecure Default Permissions, Insecure inherited permissions, Insecure preserved inherited permissions, Insecure execution-assigned permissions, Fails poorly due to insufficient permissions, Permission preservation failure, Ownership errors, Unverified Ownership, Access Control List (ACL) errors, User management errors

12. Handler Errors [HAND] (4 types)

    These categories, which are not very mature, cover various ways in which "handlers" are improperly applied to data. The specific types in this group are: Handler errors, Missing Handler, Dangerous handler not cleared/disabled during sensitive, Raw Web Content Delivery, File Upload of Dangerous Type

13. User Interface Errors [UI] (7 types)

    These categories cover WIFFs in a product's user interface that lead to insecure conditions. The specific types in this group are: Product UI does not warn user of unsafe actions, Insufficient UI warning of dangerous operations, User interface inconsistency, Unimplemented or unsupported feature in UI, Obsolete feature in UI, The UI performs the wrong action, Multiple Interpretations of UI Input, UI Misrepresentation of Critical Information

14. Interaction Errors [INT] (7 types)

    These categories cover WIFFs that only occur as the result of interactions or differences between multiple products that are used in conjunction with each other. The specific types in this group are: Multiple Interpretation Error (MIE), Extra Unhandled Features, Behavioral Change, Expected behavior violation, Unintended proxy/intermediary, HTTP response splitting, HTTP Request Smuggling

15. Initialization and Cleanup Errors [INIT] (6 types)

    These categories cover incorrect initialization. The specific types in this group are: Insecure default variable initialization, External initialization of trusted variables or values, Non-exit on Failed Initialization, Missing Initialization, Incorrect initialization, Incomplete Cleanup.

16. Resource Management Errors [RES] (11 types)

    These categories cover ways in which a product does not properly manage resources such as memory, CPU, network bandwidth, or product-specific objects. The specific types in this group are: Memory leak, Resource leaks, UNIX file descriptor leak, Improper resource shutdown, Asymmetric resource consumption (amplification), Network Amplification, Algorithmic Complexity, Data Amplification, Insufficient Resource Pool, Insufficient Locking, Missing Lock Check

17. Numeric Errors [NUM] (6 types)

    These categories cover WIFFs that involve erroneous manipulation of numbers. The specific types in this group are: Off-by-one Error, Integer Signedness Error (aka "signed integer" error), Integer overflow (wrap or wraparound), Integer underflow (wrap or wraparound), Numeric truncation error, Numeric Byte Ordering Error

18. Authentication Error [AUTHENT] (12 types)

    These categories cover WIFFs that cause authentication mechanisms to fail. The specific types in this group are: Authentication Bypass by Alternate Path/Channel, Authentication bypass by alternate name, Authentication bypass by spoofing, Authentication bypass by replay, Man-in-the-middle (MITM), Authentication Bypass via Assumed-Immutable Data, Authentication Logic Error, Missing Critical Step in Authentication, Authentication Bypass by Primary WIFF, No Authentication for Critical Function, Multiple Failed Authentication Attempts not Prevented, Miscellaneous Authentication Errors

19. Cryptographic errors [CRYPTO] (13 members)

    These categories cover problems in the design or implementation of cryptographic algorithms and protocols, or their misuse within other products. The specific types in this group are: Plaintext Storage of Sensitive Information, Plaintext Storage in File or on Disk, Plaintext Storage in Registry, Plaintext Storage in Cookie, Plaintext Storage in Memory, Plaintext Storage in GUI, Plaintext Storage in Executable, Plaintext Transmission of Sensitive Information, Key Management Errors, Missing Required Cryptographic Step, Weak Encryption, Reversible One-Way Hash, Miscellaneous Crypto Problems

20. Randomness and Predictability [RAND] (9 types)

    These categories cover WIFFs in security-relevant processing that depends on sufficient randomness to be effective. The specific types in this group are: Insufficient Entropy, Small Space of Random Values, PRNG Seed Error, Same Seed in PRNG, Predictable Seed in PRNG, Small Seed Space in PRNG, Predictable from Observable State, Predictable Exact Value from Previous Values, Predictable Value Range from Previous Values

21. Code Evaluation and Injection [CODE] (4 types)

    These categories cover WIFFs in components that process and evaluate data as if it is code. The specific types in this group are: Direct Dynamic Code Evaluation, Direct Static Code Injection, Server-Side Includes (SSI) Injection, PHP File Inclusion

22. Error Conditions, Return Values, Status Codes [ERS] (4 types)

    These categories cover WIFFs that occur when a product does not properly handle rare or erroneous operating conditions. The specific types in this group are: Unchecked Error Condition, Missing Error Status Code, Wrong Status Code, Unexpected Status Code or Return Value

23. Insufficient Verification of Data [VER] (7 types)

    These categories cover WIFFs in which the source and integrity of incoming data are not properly verified. The specific types in this group are: Improperly Verified Signature, Use of Less Trusted Source, Untrusted Data Appended with Trusted Data, Improperly Trusted Reverse DNS, Insufficient Type Distinction, Cross-Site Request Forgery (CSRF), Other Insufficient Verification

24. Modification of Assumed-Immutable Data [MAID] (2 types)

    These categories cover WIFFs in which data that is assumed to be immutable by a product, can be modified by an attacker. The specific types in this group are: Web Parameter Tampering, PHP External Variable Modification

25. Product-Embedded Malicious Code [MAL] (7 types)

    These categories cover WIFFs for intentionally malicious code that has been introduced into a product sometime during the software development lifecycle. The specific types in this group are: Back Door, Back Door, Developer-Introduced Back Door, Outsider-Introduced Back Door, Hidden User-Triggered Functionality, Logic Bomb, Time Bomb

26. Common Attack Mitigation Failures [ATTMIT] (3 types)

    These categories cover certain design problems that are more frequently known by the attacks against them. The specific types in this group are: Insufficient Replay Protection, Susceptibility to Brute Force Attack, Susceptibility to Spoofing

27. Containment errors (container errors) [CONT] (3 types)

    These categories cover WIFFs that involve the storage or transfer of data outside of its logical boundaries. The specific types in this group are: Sensitive Entity in Accessible Container, Sensitive Data Under Web Root, Sensitive Data Under FTP Root

28. Miscellaneous WIFFs [MISC] (7 types)

    These categories do not fit cleanly within any of the other main categories. The specific types in this group are: Double-Free Vulnerability, Incomplete Internal State Distinction, Other Types of Truncation Errors, Signal Errors, Improperly Implemented Security Check for Standard, Misinterpretation Error, Business Rule Violations or Logic Errors

6. ACKNOWLEDGMENTS

The work contained in this paper was funded by DHS NCSD.

7. REFERENCES

[1] "The Common Vulnerabilities and Exposures (CVE) Initiative," MITRE Corporation, (http://cve.mitre.org).

[2] "The Software Assurance Metrics and Tool Evaluation (SAMATE) project," National Institute of Science and Technology (NIST), (http://samate.nist.gov).

[3] Code Assessment Methodology Project (CAMP), part of the Protection of Vital Data (POVD) effort, Concurrent Technologies Corporation, (http://www.ctc.com).

[4] "The Preliminary List Of Vulnerability Examples for Researchers (PLOVER)," MITRE Corporation, (http://cve.mitre.org/docs/plover/).

[5] Householder, A. D., Seacord, R. C., "A Structured Approach to Classifying Security Vulnerabilities," CMU/SEI-2005-TN-003, January 2005.

[6] Leek, T., Lippmann, R., Zitser, M., "Testing Static Analysis Tools Using Exploitable Buffer Overflows From Open Source Code," Foundations of Software Engineering December, 2005 Newport Beach, CA.

[7] Waters, J. K., "Don't Let Your Applications Get You Down," Application Development Trends, July 2005.

[8] Wang, C., Wang, H., "Taxonomy of Security Considerations and Software Quality," Communications of the ACM, June 2003, Vol. 46. No. 6.

[9] Plante, A., "Beefed up OWASP 2.0 introduced at BlackHat," SearchSecurity.com, 28 July, 2005.

[10] Viega, J., "Security, Problem Solved?," QUEUE, June 2005.

[11] Ball, T., Das, M., DeLine, R., Fahndrich, M., Larus, J. R., Pincus, J., Rajamani, S. K., Venkatapathy, R., "Righting Software," IEEE Software, May/June 2004.

[12] Ranum, M. J., "SECURITY, The root of the problem," QUEUE, June 2004.

[13] Messier, M., Viega, J., "It's not just about the buffer overflow," QUEUE, June 2004.

[14] Weber, S., Karger, P. A., Paradkar, A., "A Software Flaw Taxonomy: Aiming Tools at Security," ACM Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05) St. Louis, Missouri, USA., June 2004.

[15] McGraw, G., Chess, B., Tsipenyuk, K., "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". "NIST Workshop on Software Security Assurance Tools, Techniques, and Metrics," November, 2005 Long Beach, CA.

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

[17] Viega, J., The CLASP Application Security Process, Secure Software, Inc., http://www.securesoftware.com, 2005.

[18] Department of Homeland Security National Cyber Security Division's "Build Security In" (BSI) web site, (http://buildsecurityin.us-cert.gov).