Being Explicit about Security Weaknesses

Robert A. Martin
MITRE Corporation

Sean Barnum
Cigital, Inc.

Steve Christey
MITRE Corporation

The secure software development community is developing a standard dictionary of the weaknesses that lead to exploitable software vulnerabilities. The Common Weakness Enumeration (CWE) and related efforts are intended to serve as a unifying language of discourse and act as a measuring stick for comparing the tools and services that analyze software for security issues. Without a common, high-fidelity description of these weaknesses, efforts to address vulnerabilities will be piecemeal at best, only solving part of the problem. Various efforts at DHS, DoD, NIST, NSA, and in industry cannot move forward in a meaningful fashion or with any hope of their efforts being aligned and integrated with each other so we can protect our networked systems starting with the source - the software development lifecycle. While the current driver for CWE is in code assessment tool analysis, we believe that CWE and its related efforts could have a broader impact. For the full set of information about CWE go to the web site [cwe.mitre.org.]

Introduction

More and more organizations want assurance that the software products they acquire and develop are free of known types of security weaknesses. High-quality tools and services for finding security weaknesses 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 software product 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 [1] being led by the National Institute of Standards and Technology (NIST), the Object Management Group (OMG) Software Assurance (SwA) Special Interest Group (SIG) [2], 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 require a common description of the underlying security weaknesses that can lead to exploitable vulnerabilities in software that they are targeted to resolve. Without such a common description, these efforts, as well as the Department of Defense's own Software and Systems Assurance efforts, cannot move forward in a meaningful fashion or be aligned and integrated with each other to provide the answers we need to secure our networked systems.

A Different Approach

Past attempts at developing this kind of effort 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, and as part of MITRE's participation in the DHS-sponsored NIST SAMATE effort MITRE investigated the possibility of leveraging the CVE initiative's experience in analyzing over 20,000 real-world vulnerabilities reported and discussed by industry and academia.

As part of the creation of the Common Vulnerabilities and Exposures (CVE) List [4] which is used as the source of vulnerabilities for the National Vulnerability Database (NVD) [5], over the last six 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 original groupings used in the development of CVE, while sufficient for that task, were too rough to be used to identify and categorize the functionality found within the offerings of the code security assessment industry. For example, in order to support the development of CVE content it is sufficient to separate the reported vulnerabilities in products into working categories like weak or bad authentication, buffer overflow, cryptographic error, denial of service, directory traversal, information leak, or cross-site scripting. However, for assessing code this granularity of classification groupings was too large and indefinite. Of the categories listed, for example, cross-site scripting and buffer overflows have many variants, all of which need to be identified when assessing code.

So to support use in code assessment additional fidelity and succinctness was needed as well as additional details and descriptive information for each of the different categories such as the effects, behaviors, and implementation details, etc. The preliminary classification and categorization work used in the development of CVE was revised to address the types of issues discussed above, resulting in the Preliminary List of Vulnerability Examples for Researchers (PLOVER) [6]. PLOVER includes CVE names for over 1,500 diverse, real-world examples of vulnerabilities. The vulnerabilities are organized within a detailed conceptual framework that enumerate the 300 individual types of weaknesses that caused the vulnerabilities. The weaknesses were simply grouped within 28 higher-level categories, each category with its associated CVE examples. PLOVER represents the first cut of a truly bottom-up effort to take real-world observed exploitable vulnerabilities that *do* exist in code, abstract them and group them into common classes representing more general potential weaknesses that *could* lead to exploitable vulnerabilities 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.

Creating a Community Effort

As part of the DoD/DHS Software Assurance Working Groups and the NIST SAMATE project, MITRE fostered the creation of a community of partners from industry, academia, and government to develop, review, use, and support a common weaknesses dictionary/encyclopedia that can be used by those looking for weaknesses in code, design, or architecture as well as those teaching and training software developers about the code, design, or architecture weaknesses that they should avoid due to the security problems they can have on applications, systems, and networks. The effort is called the Common Weakness Enumeration (CWE) initiative. The work from PLOVER became the major source of content for draft one of the CWE dictionary.

An important element of the CWE initiative is to be transparent to all on what we are doing, how we are doing it, and what we are using to develop the CWE dictionary. We believe this transparency is important during the initial creation of the CWE dictionary so that all of the participants in the CWE community will feel comfortable with the end result and won’t be hesitant about incorporating CWE into what they do. Figure 1 shows the overall CWE context and community involvement of the effort. We believe the transparency should also be available participants and users that will come after the initial CWE dictionary are available on the CWE Web site [7] so all of the publicly available source content is being hosted on the site for anyone to review or use for their own research and analysis.

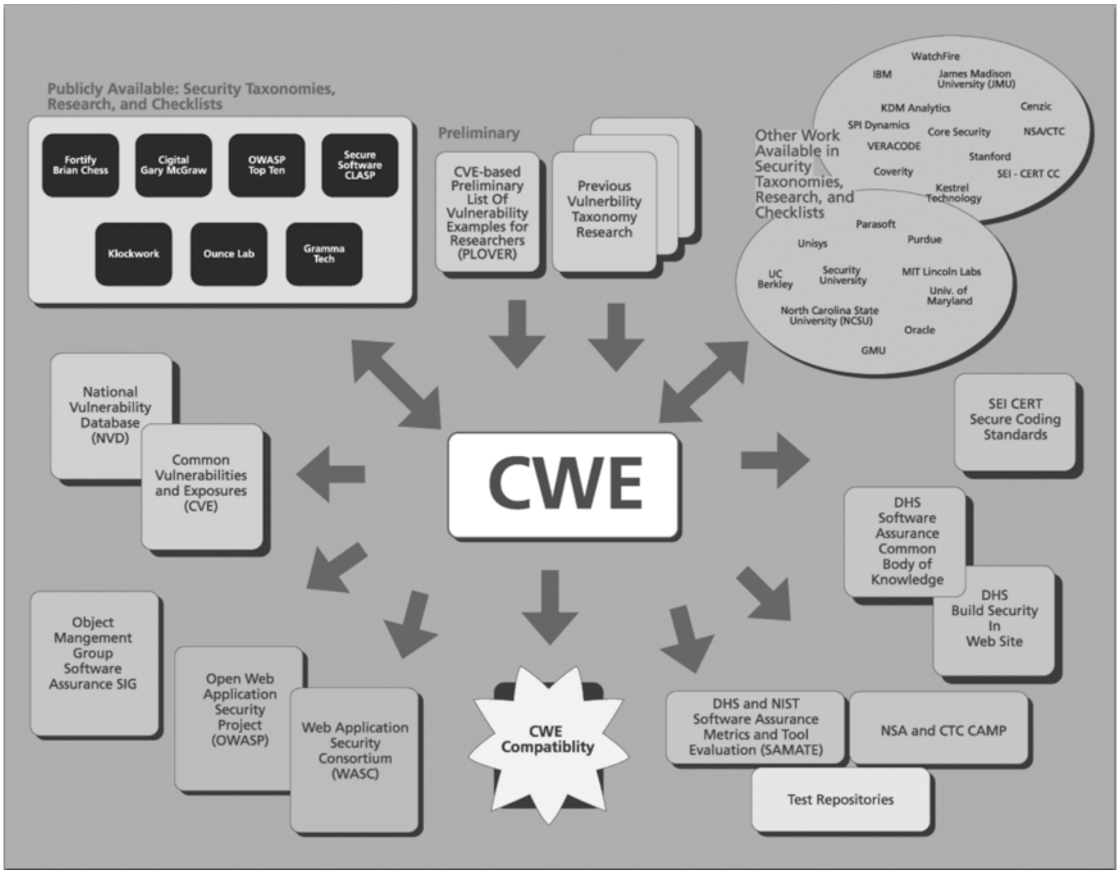

Publicly Available: Security Taxonomies, Research, and Checklists
Fortify Brian Chess
Cigital Gary McGraw
OWASP Top Ten
Secure Software CLASP
Klockwork
Ounce Lab
Gramma Tech
Preliminary
CVE-based Preliminary List Of Vulnerability Examples for Researchers (PLOVER)
Previous Vulnerbility Taxonomy Research
Other Work Available in Security Taxonomies, Research, and Checklists
WatchFire
IBM
James Madison University (JMU)
KDM Analytics
Cenzic
SPI Dynamics
Core Security
NSA/CTC
VERACODE
Stanford
Coverity
SEI - CERT CC
Kestrel Technology
Parasoft
Unisys
Purdue
UC Berkley
Security University
MIT Lincoln Labs
Univ. of Maryland
North Carolina State University (NCSU)
Oracle
GMU
SEI CERT Secure Coding Standards
National Vulnerability Database (NVD)
Common Vulnerabilities and Exposures (CVE)
CWE
DHS Software Assurance Common Body of Knowledge
DHS Build Security In Web Site
Object Mangement Group Software Assurance SIG
Open Web Application Security Project (OWASP)
Web Application Security Consortium (WASC)
CWE Compatiblity
DHS and NIST Software Assurance Metrics and Tool Evaluation (SAMATE)
NSA and CTC CAMP
Test Repositories

Figure 1: The CWE Effort's Context & Community

Currently over forty two organizations, shown in the Table, are participating in the creation and population of the CWE dictionary.

- AppSIC, LLC.
- Aspect Security
- Cenzic Inc.
- CERIAS/Purdue University
- CERT/CC
- Cigital, Inc.
- Code Scan Labs
- Core Security Technologies
- Coverity, Inc.
- Fortify Software Inc.
- International Business Machines
- Interoperability Clearing House (ICH)
- James Madison University
- Johns Hopkins University Applied Physics Laboratory
- KDM Analytics
- Kestrel Technology
- Klocwork Inc.
- Microsoft Corporation
- MIT Lincoln Labs
- MITRE Corporation
- National Institute of Standards and Technology (NIST)
- National Security Agency
- North Carolina State University
- Object Management Group
- Open Web Application Security Project (OWASP)
- Oracle Corporation
- Ounce Labs, Inc.
- Palamida
- Parasoft Corporation
- proServices Corporation
- Secure Software, Inc.
- Security Innovation, Inc.
- Security University
- Semantic Designs, Inc.
- SofCheck, Inc.
- SPI Dynamics, Inc.
- Unisys
- VERACODE
- Watchfire Corporation
- Web Application Security Consortium (WASC)
- Whitehat Security, Inc.

Table: The Common Weakness Enumeration Community

Kick Starting a Dictionary

To continue the creation of the CWE dictionary we brought together as much public content as possible, using three primary sources:

- PLOVER [6], which produced about 300 weakness concepts;
- Comprehensive, Lightweight Application Security Process (CLASP) from Secure Software, which yielded over 90 weakness concepts [8], and

- o Fortify's Seven Pernicious Kingdoms papers, which contributed over 110 weakness concepts [9].

Working from these collections as well as those contained in the thirteen other publicly available information sources listed on the CWE Web site "Sources" page we developed the first draft of the CWE List, which entailed almost 500 separate weaknesses. It took approximately six months to move from what we created in PLOVER to the first draft of CWE. The CWE content is captured in an XML document and follows the CWE schema. Two months later we updated CWE to draft 2 by cleaning up the names of items, reworking the structure, and filling in the descriptive details for many more of the items. The first change to the CWE schema came about with the addition of language and platform ties for weaknesses and the addition of specific CWE-IDs for each weakness.

Covering What Tools Find

While the third draft of CWE continued expanding the descriptions and improving the consistency and linkages, subsequent drafts will incorporate the specific details and descriptions of the 16 organizations that have agreed to contribute their intellectual property to the CWE Initiative. Under Non-Disclosure Agreements with MITRE, which allow the merged collection of their individual contributions to be publicly shared in the CWE List, AppSIC, Cenzec, Core Security, Coverity, Fortify, Interoperability Clearinghouse, Klocwork, Ounce Labs, Parasoft, proServices Corporation, Secure Software, Security Innovation Inc., SofCheck, SPI Dynamics, Veracode, and Watchfire are all contributing their knowledge and experience to building out the CWE dictionary. The first draft of CWE to include details from this set of information sources is draft 4.

Draft 5 of CWE encompasses over 600 nodes with specific details and examples of weaknesses for many of the entries. Figure 2 shows the transition from PLOVER, to CWE drafts 1 through 5, and the content structure changes that occurred during the revisions. While the initial transition from PLOVER to CWE took six months, each subsequent updated draft has occurred on a bimonthly basis.

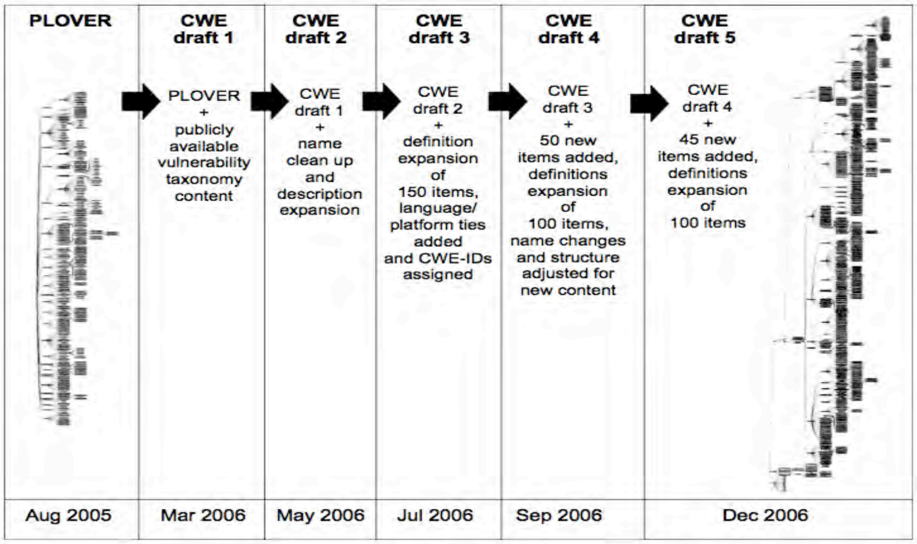
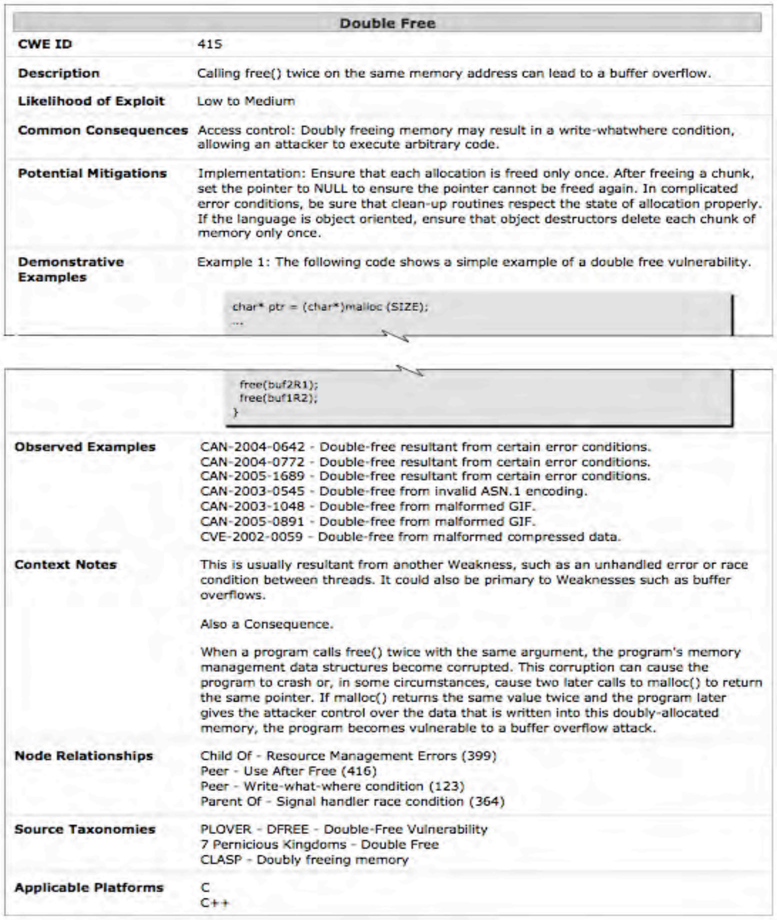

PLOVER
CWE draft 1
CWE draft 2
CWE draft 3
CWE draft 4
CWE draft 5
PLOVER + publicly available vulnerability taxonomy content
CWE draft 1 + name clean up and description expansion
CWE draft 2 + definition expansion of 150 items, language/ platform ties added and CWE-IDs assigned
CWE draft 3 + 50 new items added, definitions expansion of 100 items, name changes and structure adjusted for new content
CWE draft 4 + 45 new items added, definitions expansion of 100 items
Aug 2005
Mar 2006
May 2006
Jul 2006
Sep 2006
Dec 2006

Figure 2: From PLOVER to CWE draft 5

In addition to the sources supplying specific knowledge from tools or analysts, we are also leveraging the work, ideas, and contributions of researchers at Carnegie Mellon's CERT/CC, IBM, KDM Analytics, Kestrel Technology, MIT Lincoln Labs, North Carolina State University, Oracle, the Open Web Application Security Project (OWASP), Security Institute, UNISYS, the Web Application Security Consortium (WASC), Whitehat Security, and any other interested parties that wish to contribute. There is also a close association with the CVE project, which will ensure that newly discovered weaknesses or variants are integrated into CWE.

The contributed materials are being merged and incorporated into several of drafts of CWE (draft 5 in December 2006 and draft 6 in February 2007), which will be available for open community comments and refinement as CWE moves forward. A major part of the future work will be refining and defining the required attributes of CWE elements into a more formal schema defining the metadata structure necessary to support the various uses of CWE dictionary. Figure 3 shows a sample of the descriptive content of an entry from CWE draft 4. This example is for the Double Free weakness, CWE-ID 415.



Double Free	
CWE ID	415
Description	Calling free() twice on the same memory address can lead to a buffer overflow.
Likelihood of Exploit	Low to Medium
Common Consequences	Access control: Doubly freeing memory may result in a write-whatwhere condition, allowing an attacker to execute arbitrary code.
Potential Mitigations	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.

Demonstrative Examples	Example 1: The following code shows a simple example of a double free vulnerability.
	char* ptr = (char*)malloc (SIZE);
	...
	free(buf2R1);
	free(buf1R2);
	}

Observed Examples	CAN-2004-0642 - Double-free resultant from certain error conditions. CAN-2004-0772 - Double-free resultant from certain error conditions. CAN-2005-1689 - Double-free resultant from certain error conditions. CAN-2003-0545 - Double-free from invalid ASN.1 encoding. CAN-2003-1048 - Double-free from malformed GIF. CAN-2005-0891 - Double-free from malformed GIF. CVE-2002-0059 - Double-free from malformed compressed data.
Context Notes	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. Also a Consequence. 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.
Node Relationships	Child Of - Resource Management Errors (399) Peer - Use After Free (416) Peer - Write-what-where condition (123) Parent Of - Signal handler race condition (364)
Source Taxonomies	PLOVER - DFREE - Double-Free Vulnerability 7 Pernicious Kingdoms - Double Free CLASP - Doubly freeing memory
Applicable Platforms	C C++

Figure 3: Entry for CWE-ID 415, Double Free

However, the CWE schema will also be driven by the need to align with and support the SAMATE and OMG SwA SIG efforts that are developing software metrics, software security tool metrics, the software security tool survey, the methodology for validating software security tool claims, and developing reference datasets for testing.

For example, a major aspect of the SAMATE project is the development and open sharing of test applications that have been salted with known weaknesses so that those that wish to see how effective a particular tool or technique is in finding that type of weakness will have readily available test materials to use. These test sets are referred to as the SAMATE Test Reference Datasets (TRDs). NIST has chosen to organize the SAMATE TRDs by CWE weakness types and will also include varying levels of complexity, as appropriate to each type of weakness, so that tools that are more or less effective in finding complex examples of a particular CWE weakness can be identified. Correct constructs, that are closely aligned to the CWEs but are correct implementations, will also be included in the TRDs to help identify the false positive effectiveness of the tools. Adding complexity descriptions to the CWE schema will allow SAMATE and CWE to continue to support each other.

The OMG's Software Assurance SIG, which is using CWEs as one type of software issue that tools will need to be able to locate within the eventual OMG Software Assurance technology approach, needs much more formal descriptions of the weaknesses in CWE to allow their technological approaches to apply. OMG's planned approach for this is to use of their Semantics of Business Vocabulary and Rules (SBVR) language to articulate formal language expressions of the different CWEs. The CWE schema will have to be enhanced to allow SBVR expressions of each CWE to be included. The CWE will house the official version of the SBVR expression of that CWE.

The CWE dictionary content is already provided in several formats and will have additional formats and views into its contents added as the CWE initiative proceeds. Currently one of the ways for viewing CWE is through the CWE content page that contains an expanding/contracting hierarchical "taxonometric" view while another is through an alphabetic dictionary. The end items in the hierarchical view are hyperlinked to their respective dictionary entries. Graphical depictions of CWE content, as well as the contributing sources, are also available on the site. Finally, the XML and XML Schema Definition (XSD) for CWE are provided for those who wish to do their own analysis/review with other tools. Dot notation representations, a standard method for encoding graphical plots of information, will be added in the future.

Finally, a process to acknowledge capabilities that incorporate CWEs has been established. This "CWE Compatibility and CWE Effectiveness" program is similar to the certification and branding program used by the CVE effort but has two distinct parts, compatibility and effectiveness. The basic stages of the compatibility program are a formalized process for capability owners to publicly declare their use of CWEs and a public documentation of how their capability fulfills the requirements for finding those CWEs. The effectiveness program, which only applies to assessment capabilities, consists of a public declaration about which CWEs a capability covers and publicly available test results showing how effective the capability is in finding those CWEs.

Additional Impact and Transition Opportunities Tied to CWE

The establishment of the CWE effort is yielding 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.

The direct impacts include:

- Providing a common language of discourse for discussing, finding, and dealing with the causes of software security vulnerabilities as they are manifested in code, design, or architecture. An outgrowth of this effort is the identification of fundamental aspects of vulnerabilities that provide a framework and vocabulary that help in systematically understanding the core characteristics of vulnerabilities that goes beyond listing all the variants. This "vulnerability theory" is, in turn, improving future drafts of CWE itself.
- Allowing 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

assortment of CWEs that a given tool would find. Software purchasers will 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.

- Enabling 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).
- Enabling government and industry to leverage this standardization in their acquisition contractual terms and conditions.

There will also be a variety of alignment opportunities, where other security related efforts and CWE can leverage each other to the benefit of both. Examples of the synergies that are possible include:

- Mapping of CWEs to CVEs. This mapping would 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.
- Creation of a validation framework for tool/service vendor claims, whether used by the purchasers themselves or through a 3rd party validation service, would be able to heavily leverage the common weakness dictionary as its basis of analysis. To support this, the community would need to define the mechanisms used to exploit the various CWEs for the purposes of helping to clarify the CWE groupings and come up with verification methods for validating the effectiveness of tools for identify the presence of CWEs in 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.
- Bidirectional alignment between the common weaknesses enumeration and the SAMATE metrics effort.
- The SAMATE software security tool and services survey effort would be able to leverage this common weaknesses dictionary as part of the capability framework to effectively and unambiguously describe various tools and services in a consistent apples-to-apples fashion.
- Mapping between the CWEs and the Common Attack Pattern Enumeration and Characterization (CAPEC) effort. This mapping would provide the users of these resources the ability to quickly identify the particular weaknesses that are targeted by various types of attacks and to better understand the context of individual weaknesses through understanding how they would typically be targeted for exploitation. In combination, these two resources offer significantly higher value than either does on its own.
- Bidirectional mapping between CWEs and Coding Rules, such as those deployed as part of the DHS NCSD "Build Security In" (BSI) Web site [10], used by tools and in manual code inspections to identify common weaknesses in software.
- Incorporation of CWE into the DHS NCSD Software Assurance (SwA) Common Body of Knowledge (CBK), hosted on the BSI Web site.
- Leveraging of the OMG technologies to articulate formal, machine-parsable definitions of the CWEs to support analysis of applications within the OMG standards-based tools and models.

Finally, there are several follow-on opportunities that are currently not being pursued but that could provide significant added value to the software security industry:

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

Conclusions

This work is already helping to shape and mature the code security assessment industry, and it promises to dramatically accelerate the use and utility of automation-based assessment capabilities for organizations and the software systems they acquire, develop, and use.

Acknowledgments

The work contained in this paper was funded by DHS NCSD and is based on the efforts of a large number of individuals, but special thanks is made for the contributions of Janis Kenderdine, Conor Harris, and David Harris.

References for Whitepaper

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

[2] "The OMG Software Assurance (SwA) Special Interest Group," (http://swa.omg.org).

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

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

[5] "National Vulnerability Database (NVD)," National Institute of Science and Technology (NIST), (http://nvd.nist.gov).

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

[7] "The Common Weakness Enumeration (CWE) Initiative," MITRE Corporation (http://cwe.mitre.org).

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

[9] 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.

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

References for session (PLOVER and CWE drafts 1-5)

[Bishop1996] “A Critical Analysis of Vulnerability Taxonomies Matt Bishop and David Bailey CSE-96-11 September 1996” [http://seclab.cs.ucdavis.edu/projects/vulnerabilities/scriv/ucd-ecs-96-11.pdf]

[Bishop2003] M. Bishop. Computer Security: Art and Science. Addison-Wesley, 2003

[blexim] “Basic Integer Overflows” blexim Phrack Issue 60, Chapter 10 [http://www.phrack.org/phrack/60/p60-0x0a.txt]

[Christey2004] “Off-by-one errors: a brief explanation” Steve Christey, Secprog and SC-L mailing list posts May 5, 2004 -- Secprog List: [http://marc.theaimsgroup.com/?l=secprog&m=108379742110553&w=2] and [http://marc.theaimsgroup.com/?l=secprog&m=108379754014251&w=2]; SC-L: [http://lists.virus.org/securecoding-0405/msg00018.html]

[Christey2005a] “Second-Order Symlink Vulnerabilities” Steve Christey Bugtraq June 7, 2005 [http://www.securityfocus.com/archive/1/401682]

[Christey2005b] “On Interpretation Conflict Vulnerabilities” Steve Christey Bugtraq November 3, 2005 [http://www.securityfocus.com/archive/1/415649]

[Clowes] A Study in Scarlet, Black Hat Briefings Asia 2001 [http://www.securereality.com.au/archives/studyinscarlet.txt]

[Colley] “Crafting Sumlinks for Fun and Profit”, Shaun Colley, Infosec Writers Library, April 12, 2004 [http://www.infosecwriters.com/texts.php?op=display&id=159]

[Corsaire] “Re:[2] Corsaire Security Advisory - Multiple vendor MIME RFC2047 encoding issue” Martin O'Neal Bugtraq September 15, 2004 [http://marc.theaimsgroup.com/?l=bugtraq&m=109551582712011&w=2]

[Crosby] “Algorithmic Complexity Attacks” (Crosby, Wallach) [http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003/index.html]

[Flake] 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]

[Fortify] Fortify Descriptions. [http://vulncat.fortifysoftware.com]

[Harnhammar] “CRLF Injection” Ulf Harnhammar Bugtraq May 7, 2002 [http://cert.uni-stuttgart.de/archive/bugtraq/2002/05/msg00079.html]

[Howard2002] “When scrubbing secrets in memory doesn't work” Michael Howard Bugtraq Nov 5, 2002 [http://cert.uni-stuttgart.de/archive/bugtraq/2002/11/msg00046.html] and [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode/html/secure10102002.asp]

[Howard2003] M. Howard and D. LeBlanc. Writing Secure Code. 2nd edition. Microsoft, 2003

[Howard2005] M. Howard, D. LeBlanc, J. Viega. 19 Deadly Sins of Software Security. McGraw-Hill/Osborne, 2005

[Iyer] “Analysis of Security Vulnerabilities”, “FSM” model ** exploit involves multiple operations on several objects; exploits must pass through “elementary activities” where each one is an opportunity for security check ** [http://www.laas.fr/IFIPWG/Workshops&Meetings/44/W1/09-Iyer.pdf]

[klog] klog, “The Frame Pointer Overwrite” September 9, 1999, in Phrack Issue 55, Chapter 8 [http://kaizo.org/mirrors/phrack/phrack55/P55-08]

[LitchfieldBU] “Buffer Underruns, DEP, ASLR and improving the Exploitation Prevention Mechanisms (XPMs) on the Windows platform” September 30, 2005 [http://www.ngssoftware.com/papers/xpms.pdf]

[McHog] G. Hoglund and G. McGraw. Exploiting Software: How to Break Code. Addison-Wesley, February 2004

[Moore] “0x00 vs ASP file upload scripts” Brett Moore July 13, 2004 [http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf]

[Muffet] A. Muffet. The night the log was forged. [http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10_05.htm]

[Newsham] Format String Attacks Tim Newsham, Guardent September 2000 [http://www.lava.net/~newsham/format-string-attacks.pdf]

[Paget] "Exploiting design flaws in the Win32 API for privilege escalation. Or... Shatter Attacks - How to break Windows." August, 2002 [http://security.tombom.co.uk/shatter.html]

[PeterW] "Cross-Site Request Forgeries (Re: The Dangers of Allowing Users to Post Images)" Peter W Bugtraq June 15, 2001 [http://cert.uni-stuttgart.de/archive/bugtraq/2001/06/msg00216.html]

[PtacekNewsham] "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" Thomas H. Ptacek, Timothy N. Newsham January 1998 [http://www.insecure.org/stf/secnet_ids/secnet_ids.pdf]

[RFP] "Poison NULL byte" Rain Forest Puppy Phrack

[SanctumX] Sanctum, "Blind XPath injection", May 19, 2004 [http://www.sanctuminc.com/pdfc/WhitePaper_Blind_XPath_Injection_20040518.pdf]

[Segal] "HTTP Request Smuggling" Ory Segal June 2005 [http://www.watchfire.com/resources/HTTP-Request-Smuggling.pdf]

[Skoll] "Re: Corsaire Security Advisory - Multiple vendor MIME RFC2047 encoding" David F. Skoll Bugtraq September 15, 2004 [http://marc.theaimsgroup.com/?l=bugtraq&m=109525864717484&w=2]

[SPI] "Web Applications and LDAP Injection" SPI Dynamics

[Stearns] B. Stearns. The Java. Tutorial: The Java Native Interface. Sun Microsystems, 2005. [http://java.sun.com/docs/books/tutorial/native1.1/]

[Viega] J. Viega and G. McGraw. Building Secure Software: How to Avoid Security Problems the Right Way, 2002

[Wagner] "GNU GCC: Optimizer Removes Code Necessary for Security" Joseph Wagner Bugtraq November 16, 2002 [http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2002-11/0257.html]

[Watts] "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit," Blake Watts [http://www.blakewatts.com/namedpipepaper.html]

[Whittaker2003] J. Whittaker and H. Thompson. How to Break Software Security. Addison Wesley, 2003

[Younan] "An overview of common programming security vulnerabilities and possible solutions" Yves Younan Student thesis August 2003 [http://fort-knox.org/thesis.pdf]

[Zalewski] "Strange Attractors and TCP/IP Sequence Number Analysis" Michal Zalewski 2001 [http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm]