Being Explicit About Security Weaknesses

Robert A. Martin
MITRE Corporation

Software acquirers want assurance that the products they are obtaining are reviewed for known types of security weaknesses. Acquisition groups in large government and private organizations are beginning to use such reviews as part of future contracts, but the tools and services for performing them are new and, until recently, there was no nomenclature, taxonomy, or standard to define their capabilities and coverage. A standard dictionary of software security weaknesses has been created by the community to serve as a unifying language of discourse and as a measuring stick for comparing tools and services.

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 ongoing efforts 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] 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, which is part of the Protection of Vital Data effort [3] conducted by Concurrent Technologies Corporation, among others. While these efforts are well placed, timely in their objectives, and will surely yield high value in the end, they require a common description of the underlying security weaknesses that can lead to exploitable vulnerabilities in the software that they are targeted to resolve. Without such a common description, these efforts, as well as the DoD'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 needed answers 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 Common Vulnerabilities and Exposures (CVE) initiative's experience in analyzing more than 20,000 real-world vulnerabilities reported and discussed by industry and academia.

As part of the creation of the CVE list [4] that is used as the source of vulnerabilities for the National Vulnerability Database [5], MITRE's CVE initiative during the last six years 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 such as weak/bad authentication, buffer overflow, cryptographic error, denial of service, directory traversal, information leak, or cross-site scripting. For assessing code, however, this granularity of classification groupings was too large and indefinite. Of the categories listed, for example, cross-site scripting actually has eight different types of issues that need to be addressed or looked for when assessing code; buffer overflow covered 10 different code constructs to look for.

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 effects, behaviors, and the implementation details. The preliminary classification and categorization work used in the development of CVE was revised to address the types of issues discussed above and the result was called the Preliminary List of Vulnerability Examples for Researchers (PLOVER) [6]. PLOVER was a document that listed more than 1,500 diverse, real-world examples of vulnerabilities, identified by their CVE name. The vulnerabilities are organized within a detailed conceptual framework that enumerates the 300 individual types of weaknesses that cause the vulnerabilities. The weaknesses were simply grouped within 28 higher-level categories with a large number of real-world vulnerability examples for each type of weakness. PLOVER represents the first cut of a truly bottom-up effort to take real-world observed, exploitable vulnerabilities that *do* exist in code, to 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 SwA 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 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

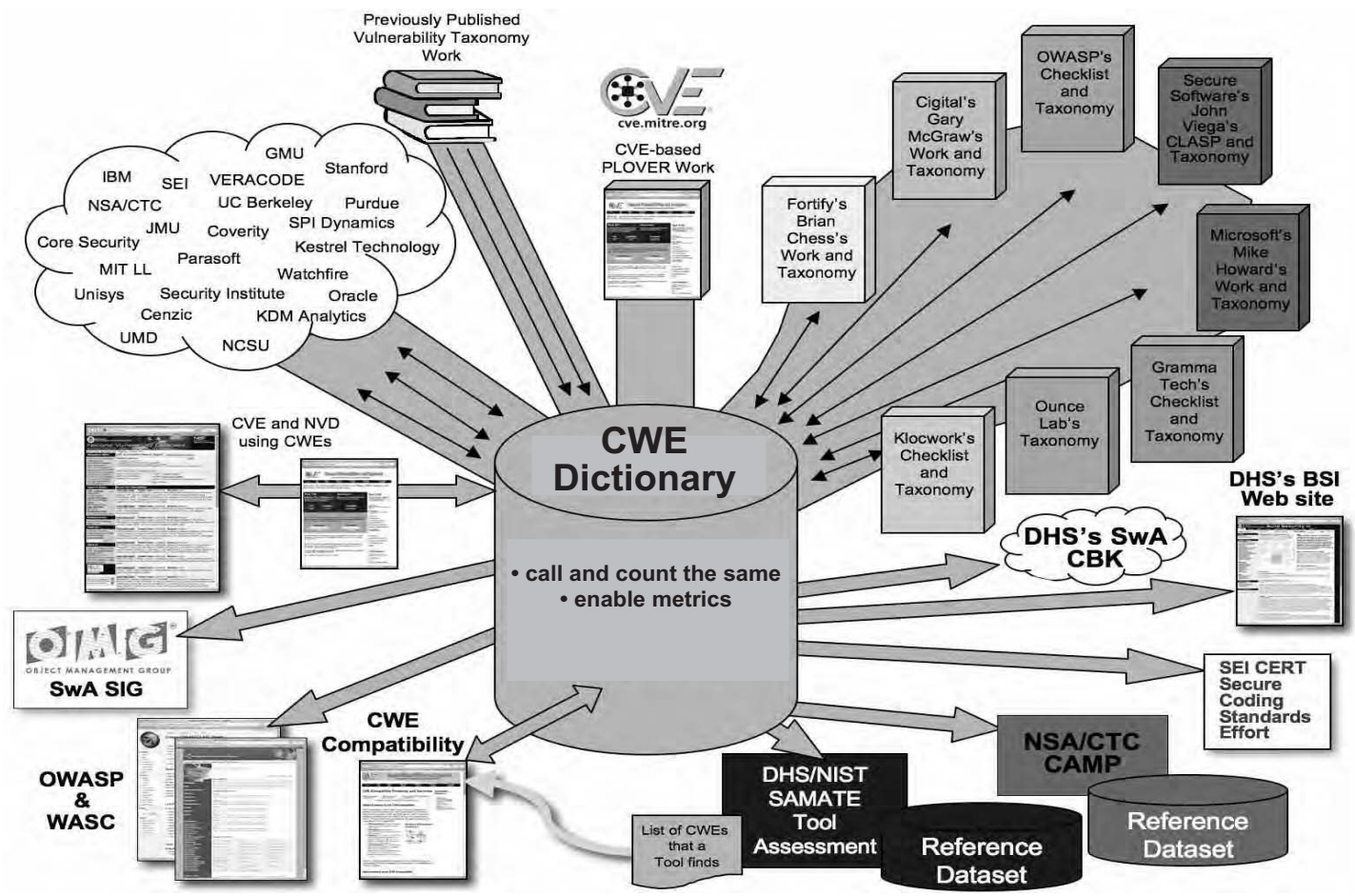


Figure 1: *The CWE Effort's Context and Community*

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 are comfortable with the end result and will not 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 to participants and users that will visit after the initial CWE dictionary is available on the CWE Web site [7]; 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.

Currently, more than 41 organizations, shown in Table 1 (see page 6), are participating in the creation and population of the CWE dictionary.

Kick-Starting a Dictionary

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

- The PLOVER collection [6] that identified more than 300 weakness types created by determining the root issues behind 1,500 of the vulnerabilities in the CVE List [4].
- The Comprehensive, Lightweight Application Security Process (CLASP) from Secure Software, which yielded more than 90 weakness concepts [8].
- The issues contained in Fortify's *Seven Pernicious Kingdoms* papers, which contributed more than 110 weakness concepts [9].

Working from these collections as well as those contained in the 13 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 with the incorporation of changes that included 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 identifications 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, Application Security Consortium, 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. Draft 4 is the first draft version to include details from this set of information sources.

Draft 5 of CWE encompasses more than 600 nodes with specific details and examples of weaknesses for many of the entries. Figure 2 shows the transition from PLOVER to CWE drafts 1-5 and the content structure changes that occurred during the revisions. While the initial transi-

• AppSIC, LLC. • Aspect Security • Cenzic, Inc. • Center for Education and Research in Information Assurance and Security/ Purdue University • Computer Emergency Response Team/ Coordination Center (CERT/CC) • Cigital, Inc. • Code Scan Labs • Core Security Technologies • Coverity, Inc. • Fortify Software, Inc. • IBM • Interoperability Clearing House • James Madison University • Johns Hopkins University Applied Physics Laboratory • KDM Analytics • Kestrel Technology • Klocwork, Inc. • Microsoft Corporation • Massachusets Institute of Technology Lincoln Labs	• MITRE Corporation • NIST • National Security Agency (NSA) • North Carolina State University • OMG • 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 1: *The CWE Community*

tion from PLOVER to CWE took six months, each subsequent updated draft has occurred on a bimonthly basis.

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's Lincoln Labs, North Carolina State University, Oracle, the OWASP, Security Institute, Unisys, the WASC, Whitehat Security, and any other interested parties that wish to contribute.

The merging and combining of the contributed materials is being incorporated into several of drafts of CWE (draft 6 in February, 2007 and draft 7 in May, 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 5. This example is for the Double Free weakness, CWE identification (ID) 415.

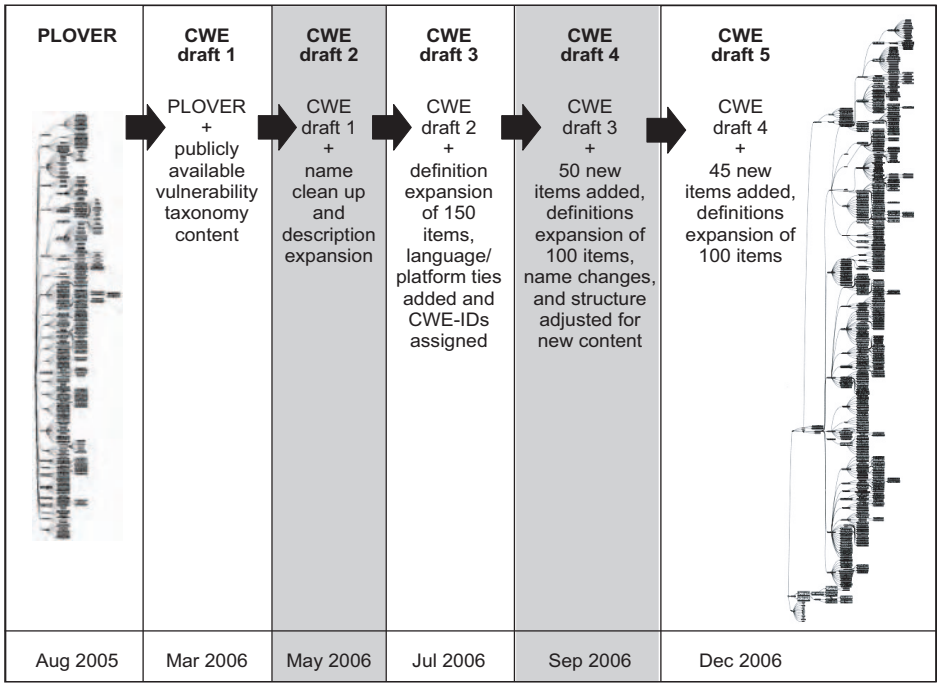
However, the CWE schema will also be driven by the need to align with and support the SAMATE and OMG SwA SIG efforts that are developing software metrics, software security tool metrics, the software security tool survey, the methodology for validating software security tool claims, and reference datasets 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 wishing to see how effective a particular tool or technique is in finding that type of weakness will have test materials readily available. These test sets are referred to as the SAMATE test reference datasets (TRDs). NIST has chosen to organize the SAMATE TRDs by CWE weakness type 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 SwA SIG, which is using CWEs as one type of software issue that tools will need to be able to locate within the eventual OMG SwA technology approach, needs more formal descriptions of the weaknesses in CWE to allow their technological approaches to apply. OMG's planned approach for this is the 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 added into its contents as the 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. Finally, the XML and XML Schema Definition (XSD) for CWE are provided for those who wish

Figure 2: *From PLOVER to CWE, Drafts 1-5*



® CERT is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

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 collection of 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 the following three types: direct impact and value, alignment with and support of other existing efforts, and enablement of new follow-on efforts to provide value that is not currently being pursued.

The direct impacts include the following:

- 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.
- 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 the following:

- Mapping of CWEs to CVEs that 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.
- Creating a validation framework for tool/service vendor claims, whether used by the purchasers themselves or through a third-party validation service, would be able to heavily leverage the common weaknesses 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 to 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.
- Establishing a bi-directional alignment between the common weaknesses enumeration and the SAMATE metrics effort.
- Using the SAMATE software security tool and services survey effort 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 effort that would

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

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++

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.

- Bi-directional mapping between CWEs and coding rules, such as those deployed as part of the DHS NCSD BuildSecurityIn Web site [10], would be used by tools and in manual code inspections to identify common weaknesses in software.
- Incorporating CWE into the DHS NCSD SwA common body of knowledge, hosted on the BuildSecurityIn 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 two follow-on opportunities that are currently not being pursued but could provide significant added value to the software security industry:

- Expansion of the coding rules catalog on the DHS BuildSecurityIn 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.

Conclusion

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 Steve Christey, Janis Kenderdine, Conor Harris, David Harris, and Sean Barnum.

References

1. NIST. "The Software Assurance Metrics and Tool Evaluation (SAMATE) Project." Jan. 2007 <http://samate.nist.gov>.
2. OMG. Object Management Group. Jan. 2007 <http://swa.omg.org>.
3. Concurrent Technologies Corporation. The Code Assessment Methodology Project. Jan. 2007 <www.ctc.com>.
4. MITRE Corporation. The Common Vulnerabilities and Exposures (CVE) Initiative. Jan. 2007 <http://cve.mitre.org>.
5. NIST. National Vulnerability Database. Jan. 2007 <http://nvd.nist.gov>.
6. MITRE Corporation. The Preliminary List of Vulnerability Examples for Researchers. Dec. 2006 <http://cve.mitre.org/docs/plover/>.
7. MITRE Corporation. The Common Weakness Enumeration Initiative <http://cwe.mitre.org>.
8. Viega, J. The CLASP Application Security Process. Secure Software, Inc., 2005 <www.securesoftware.com>.
9. McGraw, G., B. Chess, and K. Tsipenyuk. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors." NIST Workshop on Software Security Assurance Tools, Techniques, and Metrics, Long Beach, CA, Nov. 2005.
10. DHS NCSD. BuildSecurityIn. Dec. 2006 <http://buildsecurityin.us-cert.gov>.

About the Author

Robert A. Martin is a principal engineer in MITRE's Information and Computing Technologies Division. For the past seven years, his efforts have been focused on the interplay of risk management, cyber security standards, critical infrastructure protection, and the use of software-based technologies and services. Martin is a member of the Association for Computing Machinery, Armed Forces Communications and Electronics Association, Institute of Electrical and Electronics Engineers (IEEE), and the IEEE Computer Society. He has a bachelor's degree and a master's degree in electrical engineering from Rensselaer Polytechnic Institute, and a master's of business degree from Babson College.

MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
Phone: (781) 271-3001
Fax: (781) 271-8500
E-mail: ramartin@mitre.org

Coming Events

April 3-5
SAS Expo 2007
Sea-Air-Space
Washington D.C.
www.sasexpo.org/2007

April 4-5
EIG 2007
Excellence in Government 2007
Washington D.C.
www2.govexec.com/EIG2007

April 4-6
ICCSA 2007
5th International Conference on Computer Science and Applications
San Diego, CA
www.conferencehome.com/iccsa.htm

April 22-26
2nd Annual Functional Sizing Summit
Vancouver, BC, Canada
www.ifpug.org/conferences/annual.htm

April 22-26
North American CACS
North American Computer, Audit, Control and Security Conference
Grapevine, TX
www.isaca.org

April 23-26
NMDAS 2007
The 2007 Nano Materials for Defense Application Symposium
San Diego, CA
www.usasymposium.com

June 18-21
2007 Systems and Software Technology Conference
Tampa Bay, FL
www.sstc-online.org

Coming Events: **Please submit conferences, seminars, symposiums, etc. that are of interest to our readers at least 90 days before registration. E-mail announcements to nicole.kentta@hill.af.mil.**