They Know Your Weaknesses – Do You?: Reintroducing Common Weakness Enumeration

Yan Wu, Bowling Green State University
Irena Bojanova, University of Maryland, Baltimore County
Yaacov Yesha, University of Maryland University College

Abstract: Knowing what makes your software systems vulnerable to attacks is critical, as software vulnerabilities hurt security, reliability, and availability of the system as a whole. The Common Weakness Enumeration (CWE), a community effort that provides the foundation for such knowledge, is not sufficient, accurate and precise enough to serve as the common language measuring stick and provide a common baseline for developers and security practitioners. In this article, we introduce the relevant body of knowledge that consolidates CWE, including the Semantic Template and Software Fault Pattern efforts, and how static analysis tools add value through CWEs. We also provide future directions, present our vision on CWE formalization, and discuss the value of CWE for not only software assurance community, but also for Computer Science.

1. Introduction to Common Weakness Enumeration (CWE)

Software weaknesses could be exploited to compromise a system's security. This is especially critical for systems such as the Department of Defense (DoD) systems, in which the amount of software is very large. Software assurance countermeasures should be applied to address anticipated attacks against a system. Such attacks are enabled by software vulnerabilities, and those countermeasures reduce those vulnerabilities or remove them[12].

Common Weakness Enumeration (CWE) [1] is a collection of software weakness descriptions that offers a way to identify and eliminate vulnerabilities in computer systems. CWE is also used to evaluate the tools and services developed for finding weaknesses in software. CWE is community-developed and maintained by MITRE Corporation [1].

A preliminary classification of vulnerabilities, attacks, and related concepts was developed by MITRE's CVE [2] team. That effort began in 2005., CWE was developed as a list of software weaknesses that is more suitable for software security assessment [14].

1.1 History of CWE

There have been several community efforts to leverage the existing large number of diverse real-world vulnerabilities. For example, an important step towards creating the needed collection of software weakness types was the establishment of the CVE (Common Vulnerabilities and Exposures) list [2] in 1999 by MITRE. Another important step from MITRE was creating the Preliminary List Of Vulnerability Examples for Researchers (PLOVER) in 2005. PLOVER includes more than 1,500 CVE names, and 290 types of software weaknesses. The organization of those vulnerabilities is based on the types of weaknesses among 290 types that cause each vulnerability [1].

The consolidation and evolution process of CWE [1] occurred during earlier efforts to classify vulnerabilities by answering three basic questions:

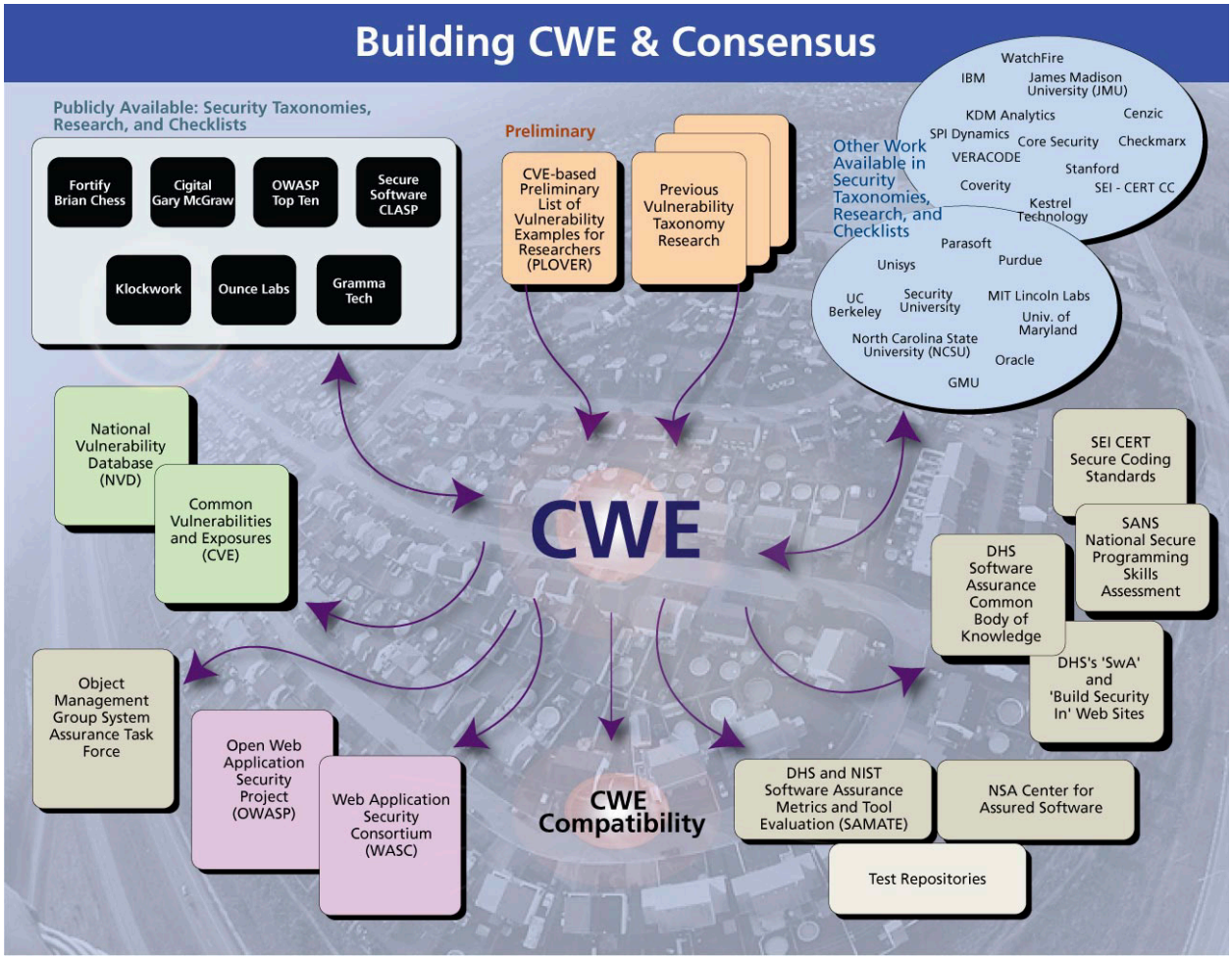
1) How did the vulnerability enter the system?
2) When did the vulnerability enter the system?
3) Where does the vulnerability appear? Or - Where is the vulnerability now?

Over a period of time, other revisions and ways to classify vulnerabilities were introduced. Until more recently, vulnerability categorizations have been developed as enumerations of weaknesses.

The CWE vision is to consolidate these efforts, and it is often compared to a "Kitchen Sink", although in a good way, as it aggregates many different taxonomies, software technologies and products, and categorization perspectives. While it provides a comprehensive record of software weaknesses, it can be a daunting task for developers to untangle the complex web of interdependencies that exist among software weaknesses captured in the CWE.

Figure 1 presents the CWE efforts context and community.

Figure 1. CWE Efforts Context and Community [http://cwe.mitre.org [1]]



1.2 CWE Concepts

Common Weakness Enumeration (CWE) [1] is a collection of descriptions of software weakness types stored as .xml, .xsd and .pdf documents. There are four major types of CWE-IDs: 1) Category, 2) Compound Element, 3) View, and 4) Weakness. The weaknesses covered by CWE have weakness IDs. Category and Compound Element are aggregations of weaknesses. Category aggregates types of weaknesses, and Compound Element aggregates a group of several events that together can result in a successful attack. View IDs are "assigned to predefined perspectives with which one might look at the weaknesses in CWE." [1]

Information provided for CWEs includes:

- CWE Identifier Number/Name of the weakness type
- Description of the type
- Alternate terms for the weakness
- Description of the behavior of the weakness
- Description of the exploit of the weakness
- Likelihood of exploit for the weakness
- Description of the consequences of the exploit
- Potential mitigations
- Node relationship information
- Source taxonomies
- Code samples for the languages/architectures
- CVE Identifier numbers of vulnerabilities for which that type of weakness exists
- References [1].

2. CWE Related Practices

Around CWE, there is a list of relevant body of knowledge such as Common Weakness Scoring System (CWSS), Common Vulnerabilities and Exposures (CVE), and Common Attack Pattern Enumeration and Classification (CAPEC). They are utilized by many institutions, including DoD, to identify and mitigate the most dangerous types of vulnerabilities in the software [12]

2.1 Use of CWE

CWE was established for those who create software, analyze software for security flaws, and provide tools and services for finding and defending against security flaws in software [1]. The CWE Compatibility and Effectiveness Program is based on six requirements: 1) "CWE Searchable," 2) "CWE Output," 3) "Mapping Accuracy," 4) "CWE Documentation," 5) "CWE Coverage," and 6) "CWE Test Results."

Meeting the first four requirements is needed for a product or a service to be designated as "CWE Compatible," and meeting all six requirements is needed for a product or ser-

vice to be designated as "CWE Effective." [1] Static analysis tools are also encouraged to map their reports to corresponding CWEs so that the results from different tools could have a standard baseline to be matched and compared.

2.2 Common Weakness Scoring System (CWSS)

The Common Weakness Scoring System (CWSS) [3] is included in CWE project. Numerically scoring software weaknesses is important, as both software developers and software consumers need to compare weaknesses in order to prioritize among various activities related to avoiding and eliminating them. CWSS enables such scoring by methods such as: Targeted, Generalized, Context-adjusted, and aggregated. CWSS 0.8 is based on the Targeted scoring method. This method is applicable to a particular package. The CWSS 0.8 scoring formula includes eighteen factors, which are divided into three groups: The Base Finding Group, the Attack Surface Group, and the Environmental Group.

2.3 Common Vulnerabilities and Exposures (CVE)

CVE is a dictionary of security vulnerabilities. It was established in 1999 in response to lack of standardization of names of vulnerabilities: different repositories could refer to the same vulnerability by a different name, resulting in difficulty in comparing software security tools.

CVE provides standard identifiers for security vulnerabilities [2], and help in finding information about a vulnerability, including ways of, and available products for, eliminating the vulnerability. It can also help in determining whether particular tools are adequate for detecting attacks that are based on particular vulnerabilities [2].

After discovering a potential security vulnerability, a CVE Numbering Authority (CNA) can assign to it a CVE identifier [2]. Then the CVE Editor posts the information on the CVE List. The Primary CNA is MITRE Corporation. Other CNAs are software vendors, (for example, Apple Inc. and Adobe Systems Incorporated), third-party coordinators, (for example, CERT/CC), or researchers (for example, Core Security Technologies). The CVE Editor is MITRE Corporation.

2.4 Common Attack Pattern Enumeration and Classification (CAPEC)

Common Attack Pattern Enumeration and Classification (CAPEC) [4] was released in 2007. It includes descriptions of attack patterns. Information provided by CAPEC is needed in the process of finding vulnerabilities in software. In order to protect against attacks, knowledge of attack patterns is valuable, in addition to knowledge of software weaknesses that can be exploited by such attacks.

3. CWE in Practice

This section describes how the static analysis tools use CWEs to tag their tool reports and why it can add value to their products.

CWE contains a fairly comprehensive collection of application architecture, design, code, and deployment errors along with mitigation advice and examples of vulnerable and correct code segments. It also describes the SANS top 25 most dangerous software errors, that often "allow attackers to completely take over the software, steal data, or prevent the software from working at all." [1]

Because of its usefulness, CWE is already recognized and adopted by many organizations. For example, 40 organizations with 71 products and services already participated in the CWE Compatibility and Effectiveness Program (http://cwe.mitre.org/compatible/organizations.html). CWE has been adopted by NIST's National Vulnerability Database (NVD) (http://nvd.nist.gov) with mappings between CVEs and CWEs, and the Open Web Application Security Project (OWASP) – Top Ten Project (https://www.owasp.org/index.php/owasp_top_ten_project). Also, as part of the NIST SAMATE project, warnings from different tools that refer to the same weakness are being matched to corresponding CWE IDs to facilitate tools evaluation [9].

State-of-the-art static analysis tools today are able to find significant types of software security weaknesses. Many tools that support CWE are accompanied by public listings of the CWEs, and they are effective at finding and tag their vulnerability reports with corresponding CWE IDs. However, some mappings are not very precise, as CWE is organized into a hierarchy and some weakness types are refinements of other weakness types; also a single vulnerability may be the result of a chain of weaknesses or the composite effect of several weaknesses. The reality is that no single tool can detect all weaknesses and multiple tools should be used for complete coverage and better they all support CWE identification to facilitate the communication among them.

Customers also ask for the mappings of found weaknesses to the CWE IDs, as this provides common grounds for evaluating tools' performance and weaknesses' coverage. Therefore, even Static Analysis Tools that claim to be responsible for only limited number of weakness types [1] should not underestimate the importance of CWE and the mappings to CWE IDs.

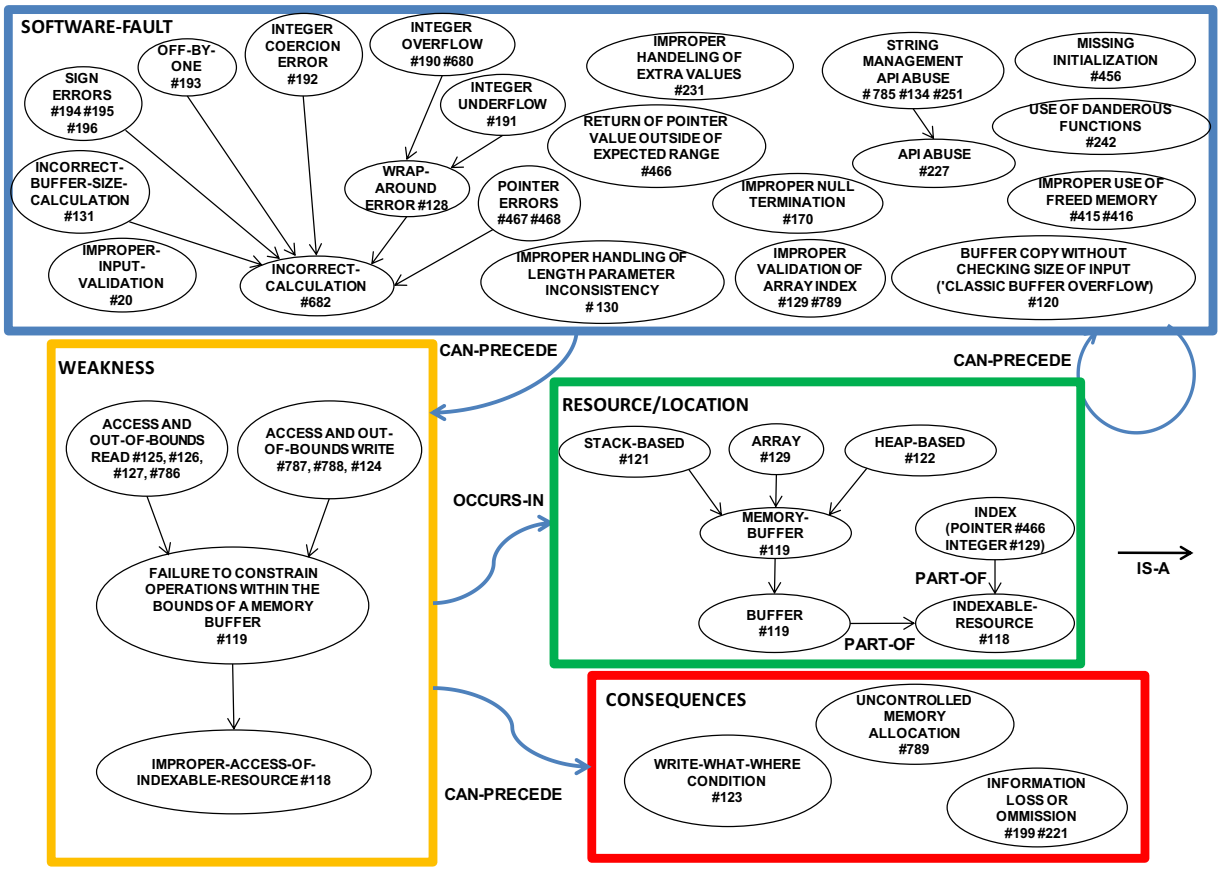
4. Improving CWE

This section describes existing efforts, which include Semantic Template and Software Fault Pattern, to improve the readability and usability of CWEs.

CWE is a collection of weaknesses with a highly tangled structure at various levels of abstraction, mixed contents of attack, behavior, feature, flaws, and all by natural language representations. It means that using its relatively unstructured weakness categories is a daunting task for stakeholders in the software development community. To help utilize the valuable contents of CWE, efforts have been made by both academia and industry to improve the readability and usability of the CWE.

Wu et. al. [5] reorganized categories of CWEs into Semantic Templates to help developers and researchers construct a more clear mental model and improve the understanding of weaknesses. To facilitate the CWE use in the study of vulnerabilities, easy-to-understand templates for each conceptually

Figure 2. Buffer Overflow Semantic Template



distinct weakness type have been developed. The templates can then be readily applied to aggregate and study project-specific vulnerability information from source code repositories.

Another approach to improve the CWE is Software Fault Patterns (SFPs) [8]. SFPs decompose CWEs by fine granularity patterns with white-box definitions, then compose them into original CWEs with invariant core and variation points. With the purpose of being integrated into a standards-based tool analysis approach, SFPs focus more on the source code faults and the features that can facilitate automation. Such automation can potentially be very valuable for software assurance activities described in [12], because CWE has an important role in those activities [12].

4.1 Semantic Templates

A Semantic Template is a human and machine understandable representation that contains the following four elements [5]:

1) Software faults that lead to a weakness
2) Resources that a weakness affects
3) Weakness characteristics
4) Consequences/failures resulting from the weakness.

The required information pieces are either expressed together within a single CWE entry or spread across multiple entries. Such complexity makes it difficult to trace the information expressed in the CWE to the information about a discovered vulnerability from multiple sources. Therefore, to facilitate CWE use in the study of vulnerabilities, easy-to-understand templates for each conceptually distinct weakness type have been developed. These templates can then be readily applied to study project-specific vulnerability information from project repositories. For example, figure 2 shows the Semantic Template for Buffer Overflow, which is an aggregation of information collected from 42 CWEs. In this Buffer Overflow Semantic Template, the four groups of relevant information were carefully collected and synthesized with "is-a" relationship inside of each group and "can-precede", "occurs-in" between the groups so that the lifecycle of a weakness from the starting point (software fault) to the end (consequences) is clearly presented.

The Semantic Templates also can provide intuitive visualization capabilities for the collected vulnerability information such as the CVE vulnerability descriptions, change history in the open source code repository, source code versions (before and after the fix), and related CAPECs [6]. Semantic Templates were shown to be helpful to programmers in constructing mental models of software vulnerabilities by an experiment described in [7]. In this experiment, 30 Computer Science students from a senior-level undergraduate Software Engineering course were selected to study six sets of vulnerability-related material with or without Semantic Templates in a pre-post randomized two-group design. The

experimental results revealed that the group with the aid of Semantic Templates could analyze vulnerabilities with shorter time and higher recall on CWE identification accuracy.

4.2 Software Fault Patterns

Software Fault Patterns (SFPs) was developed by KDM Analytics Inc. By identifying and developing white box definitions for SFPs as a formalization process, they could be integrated into a standards-based tool analysis approach, benefiting both real-time embedded and enterprise software assurance systems. Those identified SFPs will be common to more than one CWE and can be used to further define CWEs [8].

The SFP is targeted at preventing cyber-attacks by collecting and managing knowledge about exploitable weaknesses and building more comprehensive prevention, detection and mitigation solutions. With the knowledge extracted from CWE taxonomy, three transformations were executed to extract common patterns and white-box knowledge, redefine existing weaknesses as specializations of the common patterns, then invariant core and variation points are identified to redefine each SFP to further represent weakness specializations [8].

KDM Analytics defines an SFP as a common pattern with one or more associated pattern rules (conditions), representing a family of faulty computations. The SFP structure is organized by the primary SFP definition which refers to the entire secondary cluster and is arranged into invariant core and variation points [8]. SFPs can map to multiple CWEs in such a way that each CWE in the family can be defined as a specialization of the SFP with its specific variations on the identified parameters. To date, 21 primary clusters, which include totally 62 secondary clusters, and 36 unique SFPs have been identified. 632 CWEs have been categorized while only 310 of them are identified as discernible CWEs. Identified SFP definitions could lead to the development of more accurate testing tools and also improve developer education and training. They also provide benefits for a possible future formalization, since for each CWE, only the variation extension to a formalized SFP is required.

As the proof of recognition of the SFP research work, CWE-888: Software Fault Pattern (SFP) Clusters was incorporated by MITRE as a view into the CWE dictionary.

Both Semantic Templates and SFPs are designed to help understand and automate the vulnerability study. While Semantic Templates emphasize mental model construction from the human perspective, with the explanation of the four main elements of a vulnerability's lifecycle, while SFP's approach focuses on the "foot-holds", which are places in the code that present the necessary conditions for vulnerabilities, with the emphasis on the computation side to aid the test cases generator's work.

5. Future Directions on Improving CWE

This section provides future directions and our vision on CWE formalization.

CWE is a unique community effort and already has been proved to be extremely useful. For example, the NIST SAMATE project has utilized CWE during the past four Static Analysis Tool Expositions (SATE), whose goal is to advance research in static analysis tools that look for security defects in source code [9]. CWE is "a unifying language of discourse and a measuring stick for comparing tools and services" [10]. It is used in a wide variety of domains by developers and testers to look for known weaknesses in the code, design, and architecture of their software products; by consumers to make informed decisions when selecting software security tools and services; by researchers to develop new approaches and tools for software testing; and by professors to teach software developers how to avoid known weaknesses on architecture, design, and code level, in order to avoid security problems on applications, systems, and networks.

CWE is meant to be "a formal" list of software weakness types [1]. However, the CWE descriptions are currently in natural language and sometimes not accurate or precise by using phrases such as "correctly perform," "intended command," "intended boundary." For example, the description summary of CWE-119 in http://cwe.mitre.org/data/definitions/119.html includes the term "intended boundary", which is too vague. It does not indicate that it is the boundary given by the formal semantics.

CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

"The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer."

While, to mitigate the vagueness of the definition as much as possible, our tentative definition of CWE-119 is: The software can access through a buffer a memory location not allocated to that buffer [11].

Therefore, the next logical step is to formalize CWE definitions, as formal approaches are less ambiguous and offer high level of accuracy. Our vision for CWE formalization and creating a system of accurate, precise definitions of CWEs, although a high-bar, is as follows:

- Revamp CWE entries towards Software Fault Patterns
- Review for accuracy existing CWE description summaries and white-box descriptions
- Analyze descriptions meaning and remove ambiguities
- Precisely define CWE entries with required accuracy
- Decide on a formal specification language
- Formalize CWE definitions
- Determine approach for validating CWE definitions
- Determine approaches for automated generation of tools for validation and verification towards particular weaknesses.

It is challenging to identify known weaknesses as well as newly discovered weaknesses, but it is challenging also to describe them in a succinct and unambiguous manner. Formalization should come in place and help further "shape and mature the code security assessment industry and dramatically accelerate the use and utility of automation-based assessment." [1]

Semantic Templates builds on CWE, and introduces a novel reorganization of CWE. One example for a potential use of Semantic Templates is for automatic change analysis. Patches provided by contributors to open source software

may introduce vulnerabilities. Semantic Templates may help in organizing knowledge about known vulnerabilities in a way that will help patch contributors to detect vulnerabilities [5] .

Once formalized the CWE definitions could be easily expressed through formal description techniques (FDT) and used as an input for generation of testing codes. This would facilitate automatic generation of more precise CWE-compatible software analysis and profiling tools for discovery of vulnerabilities or prioritizing vulnerabilities in terms of threats and impacts. Especially valuable would be the application for generation of dynamic analysis tools, which are better at discovering run-time vulnerabilities that cannot be captured with static-code analysis techniques – for example, buffer overflow lends itself to such dynamic analysis.

6. Conclusion

CWE provides common terminology for software developers, security experts, researchers, and customers to discuss software vulnerability in design, systems architecture, and source code. Software is central to computer science and as one of the purposes of CWE is to help avoid and eliminate software flaws in various stages of software production, CWE is of value not only to the software assurance community, but to computer science as a whole.

Improving quality of software development to reduce instances of weaknesses takes work from language designers, compiler writers, educators, assurance tool developers, researchers, vulnerability trackers, software engineers, and many more. If people in these roles disagree about what constitutes a particular weakness, or even whether it is a weakness at all, communication would be difficult at best. Therefore, broadly accepted definitions should be developed to allow diverse groups to work effectively together. It is important the definitions to be unambiguous and complete to allow professional in the field to understand precisely what different software assurance tools, services, technologies, or methods can detect, mitigate, or prevent. Pure formalization of CWE would allow automatic generation of software components and tools to test for weaknesses that lead to exploitable vulnerabilities in software, create wrappers to filter out attacks that exploit them, or even rewrite the code to eliminate them.

Once precisely defined, CWEs could be formally described using a specification language such as Alloy (http://alloy.mit.edu/alloy). At its core, Alloy has a simple but expressive logic based on the notion of relations. Its syntax is designed to make it easy to build models incrementally and it has a rich sub-type facility for factoring out common features and a uniform and powerful syntax for navigation expressions.

To provoke further thinking and discussions throughout the Software Assurance community and beyond, we pose the following questions:

- What other formal methods can be used to help formalize CWEs with required accuracy and precision and at the same time allow for further extensions?
- To what granularity should CWEs be formalized? Finer granularity means more flexibility (especially when new weaknesses are identified, the extracted commonalities can reduce the re-invent work) but more effort to create them; Coarser granularity indicates the easy-to-use weakness items while we need to re-invent the wheel every time.
- How can the formalized CWEs be used and in which domains? For education and training? To prevent vulnerabilities? To integrate into software IDEs, test tools, and tools that generate test tools? To integrate in application security and development security technical implementation guides such as that of DOD [13].
- How can an automatic system be constructed to record newly identified vulnerabilities and classify them by CWEs? With better formalization and finer granularity of CWE definitions (which also means limited dictionary for weaknesses, better taxonomy of vulnerabilities), text mining could be the potential technique to mapping CVEs to CWEs at least semi-automatically.

REFERENCES

1. MITRE. "CWE Common Weakness Enumeration." http://cwe.mitre.org
2. MITRE. "CVE Common Vulnerabilities and Exposure." http://cve.mitre.org
3. MITRE. "CWE Common Weakness Enumeration Common Weakness Scoring System (CWSS) CWSS Version 0.8." June 2011. Project Coordinator: Bob Martin, Document Editor: Steve Christey. http://cwe.mitre.org/cwss
4. MITRE. "Common Attack Pattern Enumeration and Classification (CAPEC)TM A Community Knowledge Resource for Building Secure Software." http://makingsecuritymeasurable.mitre.org/docs/capec-intro-handout.pdf
5. Y. Wu, R. A. Gandhi, and H. Siy. "Using semantic templates to study vulnerabilities recorded in large software repositories." 2010 ICSE Workshop on Software Engineering for Secure Systems, SESS '10, pages 22-28, New York, NY, USA, 2010. ACM.
6. R. Gandhi, H. Siy, Y. Wu. "Studying Software Vulnerabilities." CrossTalk, The Journal of Defense Software Engineering, September/October 2010.
7. Y. Wu, H. Siy, R. Gandhi. "Empirical Results on the Study of Software Vulnerabilities (NIER Track)." 33rd International Conference on Software Engineering (ICSE 2011), Honolulu, Hawaii. May 2011.
8. B.A. Calloni, D. Campara, and N. Mansourov. (2011). Embedded Information Systems Technology Support (EISTS) ---Task Order 0006: Vulnerability Path Analysis and Demonstration (VPAD), Volume 2 - White Box Definitions of Software Fault Patterns.
9. NIST. "Special Publication 500-297 Report on the Static Analysis Tool Exposition (SATE) IV." http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-297.pdf
10. R. Martin, S. Barnum, S. Christey. "Being Explicit about Security Weaknesses" http://cwe.mitre.org/documents/being-explicit/BlackHat_BeingExplicit_WP.pdf
11. Paul E. Black, Yan Wu, Yaacov Yesha, Irena Bojanova, in preparation.
12. Deputy Assistant Secretary of Defense for Systems Engineering (DASD(SE)) and Department of Defense Chief Information Officer (DoD CIO). Software Assurance Countermeasures in Program Protection Planning. Washington, D.C. 2014. www.acq.osd.mil/se/docs/SwA-CM-in-PPP.pdf
14. DISA for DOD, Application Security and Development Security Technical Implementation Guide (STIG), Version 3, Release 8. 25 July2014, http://iase.disa.mil/stigs/app-security/app-security/Pages/index.aspx
15. MITRE: CWE Common Weakness Enumeration, Frequently Asked Questions (FAQ), http://cwe.mitre.org/about/faq.html#A.8

ABOUT THE AUTHORS

Yan Wu is currently working as an assistant professor at Computer Science Department of Bowling Green State University, and she previously was a guest researcher in SAMATE team at NIST. She received her Ph.D. degree in Information Technology in 2011 from the University of Nebraska at Omaha. The main goal of her research is to conduct empirical study on analyzing software engineering knowledge in order to support the development and maintenance of reliable software-intensive systems.

E-mail: yanwu@bgsu.edu

Irena Bojanova is a professor and program director of Information and Technology Systems at UMUC. She is the founding chair of the IEEE CS Cloud Computing STC, a general chair of the IT Professional Conference <http://tinyurl.com/itproconf> , and coeditor of Encyclopedia of Cloud <http://tinyurl.com/EncyclopediaCC> Computing (Wiley, to appear in 2014). She is also an associate editor in chief of IT <http://www.computer.org/itpro> Professional and an associate editor of IEEE Transactions on Cloud Computing <http://www.computer.org/portal/web/tcc> . You can read her cloud computing blog at www.computer.org/ portal/web/Irena-Bojanova.

E-mail: irena.bojanova@umuc.edu

Yaacov Yesha is a Professor at the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County. He received his PhD in Computer Science in 1979 from the Weizmann Institute of Science. He has received substantial research funding from government and industry. He was a program committee member of several conferences and a Chair of two workshops at IBM CASCON 2007.

E-mail: yayesha@cs.umbc.edu