Empirical Results on the Study of Software Vulnerabilities (NIER Track)

Yan Wu, Harvey Siy, Robin Gandhi
College of Information Science and Technology
University of Nebraska at Omaha
Omaha, Nebraska, USA
{ywu, hsiy, rgandhi}@unomaha.edu

ABSTRACT
While the software development community has put a significant effort to capture the artifacts related to a discovered vulnerability in organized repositories, much of this information is not amenable to meaningful analysis and requires a deep and manual inspection. In the software assurance community a body of knowledge that provides an enumeration of common weaknesses has been developed, but it is not readily usable for the study of vulnerabilities in specific projects and user environments. We propose organizing the information in project repositories around *semantic templates*. In this paper, we present preliminary results of an experiment conducted to evaluate the effectiveness of using semantic templates as an aid to studying software vulnerabilities.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management—*life cycle*; H.3.4 [**Information Systems**]: Information Storage and Retrieval—*systems and software*

General Terms
Reliability, Experimentation, Security.

Keywords
Software vulnerability, repository, buffer overflow, experiment.

1. INTRODUCTION
With new software vulnerabilities discovered everyday, a systematic study of their characteristics is a subject of immediate need. Most software development projects dedicate some effort to document, track and study reported vulnerabilities. This information is recorded in existing project repositories such as log of changes in version control systems, entries in bug tracking systems and communication threads in mailing lists. As these repositories were created for different purposes, it is not straightforward to extract vulnerability-related information. In large projects, these repositories store vast amounts of data in which the relevant information is buried. Natural language descriptions and discussions do not facilitate mechanisms to aggregate vulnerability artifacts from multiple sources or pinpoint the actual software fault and affected software elements. While a significant body of knowledge exists for categorizing software weaknesses, it is hardly applied in the context of a software project. Little or no effort has been made to improve the mental model of the software developer to sense the possibility of vulnerability with growing software complexity.

We are faced with two problems: *information overload* in software repositories and, paradoxically, *lack of information* or security know how among project stakeholders. This condition is compounded by the fact that the complete record of information is scattered over several separate systems with different information schemas and natural language descriptions.

We propose organizing the information in project repositories around semantic templates [7]. We define semantic templates to be generalized patterns of relationship between software elements, and their association with known high level phenomena in the security domain. Further information can be found in our earlier work [7][18][8]. Currently, semantic templates for several major classes of vulnerability in Apache web server [14] (e.g., buffer overflow, injection, etc.) have been developed.

Through a rigorous knowledge engineering process the primary abstractions of the templates are derived from the Common Weakness Enumeration (CWE) [6], a community-driven taxonomy of software weaknesses. Next, for known vulnerabilities, its related artifacts in a project repository and descriptions of the corresponding fixes are tagged with concepts from the template. Based on the characteristics of the resources affected by these vulnerabilities, other similar resources in the software can be identified for closer inspection and verification. In earlier work, we have described our methodology for systematically constructing semantic templates [7] as well as annotation of vulnerability artifacts [18].

In this paper, we describe preliminary results from an empirical study to evaluate the effectiveness of using semantic templates. In this experiment, we used several reported vulnerabilities in the Apache Web Server [14]. We extracted multiple artifacts concerning these vulnerabilities from the project's repositories, such as the project's public website, version control system, bug tracking system, as well as from the National Vulnerability Database [12]. During the experiment, subjects reviewed these artifacts, some with the aid of semantic templates and others without. They are then asked a series of questions to test their understanding of the reported vulnerabilities. In our preliminary analysis of results, we find statistically significant differences in the identification of CWE weakness categories necessary to explain a recorded vulnerability with and without the templates.

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

Also, subjects using semantic templates arrived at the answers much sooner than those using only the CWE document. These results encourage confidence in the effectiveness of semantic templates to study software vulnerabilities.

2. BACKGROUND AND RELATED WORK

2.1 Vulnerability Categorization

Software developers can learn a lot from past incidents. Given the volume and rapid rate at which new vulnerabilities are discovered, it was recognized fairly early in the security community that some sort of classification and categorization would be required to generate useful insights [1][3]. The software flaw taxonomy [11] was one of the early efforts to classify vulnerabilities by asking three basic questions: 1) How did it enter the system; 2) When did it enter the system and 3) Where in the system is it manifest. Over a period of time several other refinements and ways to classify vulnerabilities were introduced [4][2]. More recently vulnerability categorizations have been developed as enumerations of weaknesses [6], ranked lists based on frequency of occurrence [15], domain-specific lists [16], or based on experience of security practitioners [9][17].

The CWE vision is to consolidate these efforts, and enable a more effective discussion, description, selection, and use of software security tools and services [6]. The CWE 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.

2.2 Semantic Templates

We introduce the notion of semantic templates to simplify the organization of CWE information. In analyzing hundreds of CWE entries, we observe that the expression of weakness classes and categories in the CWE often mix (a) the characteristics of a weakness, (b) its preceding software faults, (c) resources/locations that the weakness occurs in, and (d) the consequences that may follow from the weakness. Thus a key feature in the development of semantic templates is the separation of these four concerns. Specifically, for each class of vulnerabilities (e.g., buffer overflow), we developed categorizations according to these four concerns. For illustration, the semantic template for buffer overflow is given in Figure 1. In this figure, we have isolated buffer overflow-related concepts into four concerns. (The numbering beneath each concept refers to CWE entries that most closely illustrate that concept.) Semantic templates annotate a reported vulnerability which can be easily encoded in an ontology. For example, a long-to-int type conversion (*fault*) of an index to a dynamically allocated buffer can lead to integer overflow (*fault*) which leads to a heap-based buffer (*resource*) overwrite (*weakness*) which causes a process to crash (*consequence*).

3. EMPIRICAL STUDY

To evaluate the effectiveness of semantic templates, we conducted an experiment. The premise is that, in large software development efforts, there are expert developers and there are novices. (Analogously in open source software development, there are core developers and there are occasional contributors.) While focused on understanding the domain and the primary functionalities of a software system, novices tend to ignore "secondary" issues such as security. Often out of ignorance or inability to perceive possible security weaknesses, novice developers may make similar kinds of mistakes that lead to vulnerabilities. We are interested in measuring how much effort it takes to study a reported vulnerability and to assess whether a relative newcomer to a project can quickly absorb the available information to make a meaningful assessment of the nature of a reported vulnerability.

3.1 Experimental Design

We designed an experiment to evaluate the ability of subjects to study a set of reported vulnerabilities with and without the semantic templates. Our null hypotheses are as follows:

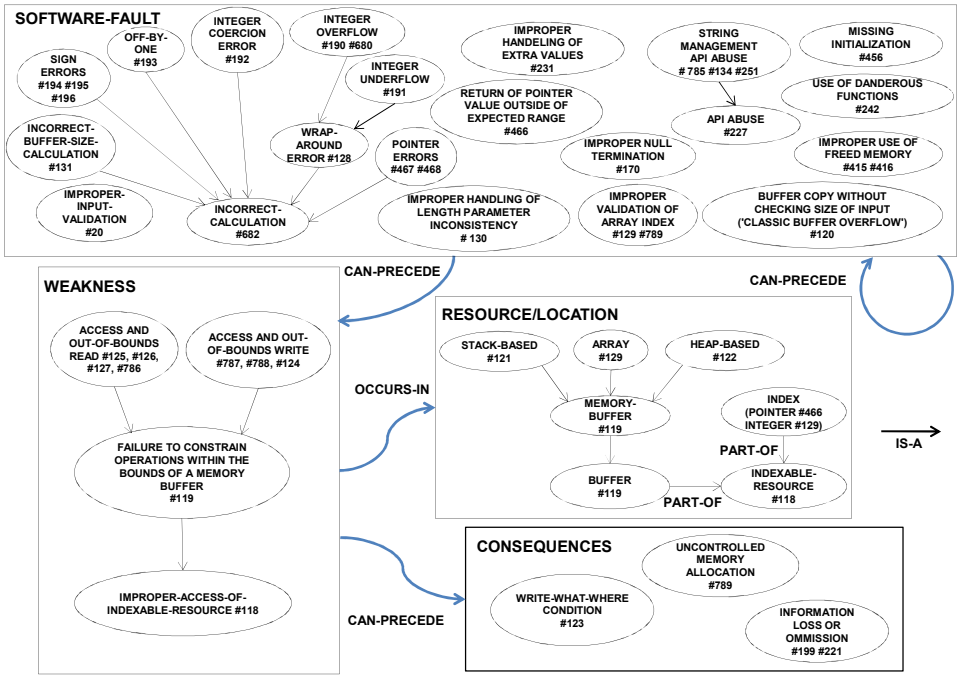

Figure 1: A Buffer Overflow Semantic Template [18]

$H1_0$ There is no reduction in completion time for subjects who use semantic templates compared to those who do not.

$H2_0$: There is no improvement in accuracy of understanding of vulnerabilities for subjects who use semantic templates compared to those who do not.

The experiment was conducted with 30 Computer Science students from a senior-level undergraduate Software Engineering course. The range of real-world software development experience varied from none to more than 5 years. The students had no prior knowledge of or experience with semantic templates.

We employed a pre-post randomized two-group design [10]. The subjects were randomly divided into 2 groups. There were 2 rounds. In Round 1, all subjects studied the vulnerability-related material with no additional aid. In Round 2, subjects in Group 2 were given semantic templates to assist in their study.

3.2 Setting and Materials

The experimental objects to be studied consisted of buffer overflow vulnerabilities reported in the Apache Web Server. We selected the buffer overflow vulnerability for the experiment because it was relatively well known (ironically, it also occurs most often). This was reinforced by a simple survey of the subjects which indicated that most have some basic understanding of buffer overflow. Of the 14 vulnerability reports that were unanimously voted as buffer overflow weaknesses by the investigators (authors of this paper); we selected 6 for the study (3 reports to study for each round). The reports were carefully chosen to account for the limited time that subjects had to examine the materials. Each round was allotted 60 minutes.

The materials given to the subjects during the experiment include the vulnerability ID (a standard identifier known as CVE [5]), descriptions from both the Apache website and the National Vulnerability Database (NVD), change description logs from the Apache Subversion repository, and the source code differences before and after the vulnerability was fixed. The subjects also had access to a hyperlinked CWE PDF document [6]. For each CVE, subjects marked start and end times, and answered the following:

Q1: List the related CWE entries for the CVE investigated.

Q2: What faults may have led to the CVE?

Q3: How does the fault lead to failure? (For example, what makes the software vulnerable? What data or data structures get corrupted? What failure conditions arise?)

Q1 is the main question of interest. The identification of CWEs would enable the use of the compiled knowledge of recurring vulnerabilities. Responses to Q2 and Q3 would provide additional insight on the effect of correct CWEs in explaining the vulnerability. These questions are primarily geared towards the investigation of a vulnerability and thus, limit the extent to which students can sense the research questions from them.

3.3 Variables

The experiment manipulated these *independent variables*:

1. **Group** - refers to the group assigned (1 or 2).
2. **Round** - refers to the experiment round (1 or 2).
3. **Vulnerability ID** - the vulnerability under study (1-1, 1-2, 1-3, 2-1, 2-2, 2-3). **1-1**: CVE-2004-0492, **1-2**: CVE-2004-0493, **1-3**: CVE-2010-0010, **2-1**: CVE-2009-0023, **2-2**: CVE-2005-1268, **2-3**: CVE-2009-2412.

These self-reported *subject variables* were collected:

1. **Programming skill level**
2. **Reading comprehension and writing skill levels** - ability to read and write technical English documents.

These *dependent variables* were collected for each vulnerability:

1. **Time to complete** - time (in minutes) to study and complete the questions about that vulnerability.
2. **CWE identification accuracy** - CWEs correctly identified as related to the vulnerability (obtained from Q1 responses and measured using precision and recall).
3. **Fault identification accuracy** - a score (scale of 1-5) on the accuracy of identification of the software fault that led to the vulnerability (from Q2 responses).
4. **Description accuracy** - a score (scale of 1-5) on the accuracy of description of the vulnerability—its manifested problem, resources impacted, and consequences (from Q3 responses).

The set of relevant CWEs for each vulnerability report was determined by consensus among the co-authors prior to the experiment. To achieve reliability of the scores for the last two measures, each co-author independently provides a score, which would allow for assessment of inter-rater reliability. Hence, the subject's accuracy of understanding vulnerabilities is measured using 3 different metrics: CWE identification accuracy, fault identification accuracy and description accuracy.

3.4 Preparation and Conduct of Experiment

To prepare for the experiment, all subjects were given a lecture on software vulnerabilities, CVE and CWE. They took part in an exercise to practice navigating the CWE relationships (e.g. parent of, peer of, etc.) to find descriptions of related weaknesses. Additionally, a demonstration was performed to walk them through the process of analyzing one vulnerability by examining its related information from the project repository.

Rounds 1 and 2 were conducted on separate days. The experiment was conducted in a supervised laboratory environment. Each subject was initially given the question sheet and related documents and information for the first vulnerability report. Upon answering all the questions for that vulnerability, the subject submitted the completed question sheet, the time is marked, and the subject proceeded to the next vulnerability. In this way, we ensured that all times reported are accurate to the minute.

Immediately before Round 2, a brief 15 minute tutorial on the use of semantic templates was provided to Group 2. The short tutorial time was chosen to accommodate the experiment within a class period of 75 minutes and to simulate a brief training available to a newcomer on the job.

4. RESULTS

We present the initial analysis of time and CWE identification. Fault identification and description accuracy will be presented in a larger publication. We used t-tests to test the hypotheses with respect to time and CWE identification precision and recall as they were all normally distributed (using the Shapiro-Wilk [13] test for normality). We used one-tailed tests as we were looking for improvements using the templates. We also performed an analysis of variance on each measure to verify that none of the other subject variables had any significant effect.

4.1 Time to Completion

The boxplots in Figure 2 depict the time it took each subject to analyze each vulnerability report. The top row shows the comparisons between the two groups for Round 1 when neither

group used semantic templates. The bottom row shows the comparisons in Round 2 with Group 2 using semantic templates.

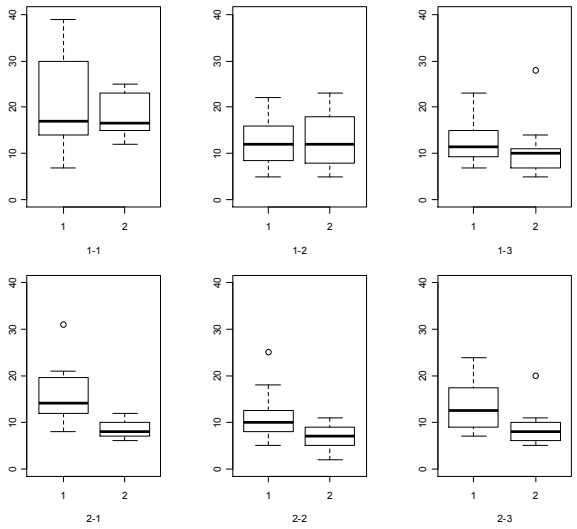

Figure 2: Time to completion (minutes) per vulnerability

Figure 2 shows that semantic templates have a noticeable effect on shortening the time to completion. This was verified using t-tests. The p-values are in Table 1 (tests with $p < 0.05$ are in bold).

Table 1: p-values of one-tailed t-tests for Time data

Round 1	(1-1) 0.3627	(1-2) 0.5855	(1-3) 0.1516
Round 2	(2-1) **0.0001**	(2-2) **0.0030**	(2-3) **0.0015**

The results show that for Round 1, there was no difference in performance between the two groups. Comparing Group 1's performance across the two rounds also shows no significant difference. Thus the results point to the use of semantic templates as the key factor leading to a decrease in time to completion. As a result, we reject $H1_0$.

4.2 CWE Identification Accuracy

The CWE identification accuracy estimates how well the subjects identified the correct set of CWEs. This was measured using precision and recall. The t-tests indicated that use of semantic templates did not have significant impact on precision, but positively affected recall. The p-values for recall are in Table 2.

Table 2: p-values of one-tailed t-tests for CWE recall

Round 1	(1-1) 0.0683	(1-2) 0.9481	(1-3) 0.2286
Round 2	(2-1) **0.0141**	(2-2) **0.0093**	(2-3) **0.0021**

The results indicate that semantic templates help in identifying the correct CWEs, although their use does not stop subjects with limited knowledge of the software project from identifying many false positives, just like the control group. In order to answer $H2_0$ with respect to CWE identification, we are also performing additional analysis to assess if there is a difference in terms of improvement in precision and recall between the two rounds.

5. DISCUSSION

Selection threats were mitigated by randomized assignment of subjects to groups. We also checked that the self-reported subject variables did not have any statistically significant contribution to the dependent variables measured. As with other experiments conducted with student subjects, generalization of results is difficult. First of all, students are not professionals and they are not familiar with the Apache product and development languages. We argue that this is not a big drawback as they would be close to the profile for a novice developer. Second, a real industrial setting would be very different from the laboratory environment. We tried to mitigate this somewhat by using a real-world project with real reported vulnerabilities.

We have presented some preliminary results on an experiment on studying software vulnerabilities. Using semantic templates reduced the time to study reported vulnerabilities and was instrumental in improving recall of relevant CWEs. Further insights will be gained through the ongoing analysis of fault identification and description data. These initial results suggest that semantic templates provide a useful approximation of the mental model to study software vulnerabilities. Furthermore, though the experiment was carried out over relatively localized vulnerabilities which do not require extensive code analysis, we believe that the difference in effects will be magnified with more complex vulnerabilities. Finally, most subjects in our study were familiar with the concept of a buffer overflow; we expect the semantic templates to perform even better when a user has little knowledge about a particular class of weakness.

6. ACKNOWLEDGMENTS

This research is funded in part by Department of Defense (DoD)/Air Force Office of Scientific Research (AFOSR), NSF Award Number FA9550-07-1-0499, under the title "High Assurance Software".

7. REFERENCES

[1] Abbott, R.P., Chin, J.S. et al. The RISOS Project. Lawrence Livermore Lab TR NBSIR-76-1041,1976.
[2] Aslam. Y. A Taxonomy of Security Faults in the UNIX Operating System. Purdue University, August 1995.
[3] Bisbey, R. and Hollingworth, D. Protection Analysis: Final Report. ARPA ORDER NO. 2223, ISI/SR-78-13 May 1978.
[4] Bishop, M. A Taxonomy of UNIX System and Network Vulnerabilities. UC Davis, CSE-95-10, May 1995.
[5] CVE - Common Vulnerabilities and Exposures. http://cve.mitre.org.
[6] CWE - Common Weakness Enumeration Version 1.6. 29 Oct. 2009. The MITRE Corporation. http://cwe.mitre.org/.
[7] Gandhi, R. A. Siy, H., and Wu, Y. Studying Software Vulnerabilities. CrossTalk, September/October 2010.
[8] Gandhi, R. A. Studying Software Vulnerabilities (companion Website). http://faculty.ist.unomaha.edu/rgandhi/st/
[9] Howard, M, LeBlanc, D., Viega. J. 19 Deadly Sins of Software Security Programming Flaws and How to Fix Them, 2005.
[10] Judd, C., et al. Research Methods in Social Relations. 1991.
[11] Landwehr, C., et al. A Taxonomy of Computer Program Security Flaws with Examples. ACM Computing Surveys 26, 3, Sept., 1994
[12] National Vulnerability Database. http://nvd.nist.gov/
[13] Shapiro, S. et al. An analysis of variance test for normality, 1965.
[14] The Apache HTTP Server Project. http://httpd.apache.org
[15] The Ten Most Critical Web Application Security Vulnerabilities. The Open Web Application Security Project (OWASP), 2007.
[16] Web Application Security Consortium 2005.
[17] Weber, S. et al. A Software Flaw Taxonomy: Aiming Tools at Security. (SESS'05) St. Louis, Missouri, June 2005.
[18] Wu,Y., Gandhi, R.A., and Siy, H. Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories. In Proc. 6th Intl. Workshop on Software Engineering for Secure Systems (SESS'10), South Africa, Cape Town. 2010.