10-2011

Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories

Yan Wu

University of Nebraska at Omaha

Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories

By

Yan Wu

A DISSERTATION

presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Information Technology

Under the Supervision of Dr. Harvey Siy and Dr. Robin A. Gandhi

Omaha, Nebraska

October, 2011

Supervisory Committee:

Dr. Parvathi Chundi, Department of Computer Science

Dr. Zhenyuan Wang, Department of Mathematics

Dr. Mansour Zand, Department of Computer Science

Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories

Yan Wu, Ph.D.

University of Nebraska, 2011

Advisors: Dr. Harvey Siy and Dr. Robin A. Gandhi

Software vulnerabilities allow an attacker to reduce a system's Confidentiality, Availability, and Integrity by exposing information, executing malicious code, and undermine system functionalities that contribute to the overall system purpose and need. With new vulnerabilities discovered everyday in a variety of applications and user environments, a systematic study of their characteristics is a subject of immediate need for the following reasons:

- *The high rate in which information about past and new vulnerabilities are accumulated makes it difficult to absorb and comprehend.*
- *Rather than learning from past mistakes, similar types of vulnerabilities are observed repeatedly.*
- *As the scale and complexity of current software grows, better mental models will be required for developers to sense the possibility for the occurrence of vulnerabilities.*

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 complicated and not readily usable for the study of vulnerabilities in specific projects and user environments. Also the discovered vulnerabilities from different projects are collected in various databases with general metadata such as dates, person, and natural language descriptions but without the links to other relevant knowledge, they are hard to be utilized for the purpose of understanding vulnerabilities.*

This research combines the information sources from these communities in a way that facilitates the study of vulnerabilities recorded in large software repositories. We introduce the notion of ***Semantic Template*** *to integrate the scattered information relevant to understand and discover vulnerabilities. We evaluate the use of semantic templates by applying it to analyze and annotate vulnerabilities recorded in software repositories from the Apache Web Server project. We refer to software repositories in a general sense that includes source code, version control data, bug reports, developer mailing lists and project development websites. We derive semantic templates from community standards such as the Common Weaknesses Enumeration (CWE) and Common Vulnerabilities and Exposures (CVE). We rely on standards in order to facilitate the adoption, sharing and interoperability of semantic templates.*

This research contributes a novel theory and corresponding mechanisms for the study of vulnerabilities in large software projects. To support these claims, we discuss our experiences and present our findings from the Apache Web Server project.

Keywords

Annotation, CWE, CVE, CAPEC, Vulnerability, Apache Web Server project, Change History Repository, Ontology, Classification, Prediction, Trend Analysis, Vulnerability Patterns, Semantic Template

Acknowledgement

First, I would like to thank Dr. Harvey Siy, whom I have worked with for over 5 years. He helped broaden my views, kept me on track and pushed me at times when I got discouraged. I also thank Dr. Robin A. Gandhi, whose collaboration brought fresh knowledge into our small team. He always can enlighten me with new concepts and ideas and provide assistance when I lose track. At the same time, I learned a lot of editing and presentation skills from him. I am lucky to have them as my dissertation co-advisors. I sincerely appreciate all their support and patience, and look forward to continuing to work with them on future research.

In addition, I would like to thank Dr. William Mahoney and the Nebraska University Center for Information Assurance (NUCIA) for sponsoring my research.

Also, I especially want to thank my husband, Li Fan, for managing the household and family needs while I spent nights and weekends working on my dissertation. And the encouragement I received from my parents and friends was the power driving me forward. Especially in the period my parents visited me, they took care of everything so that I can focus on my dissertation work. I would also like to thank the members of my Dissertation committee, Dr. Parvathi Chundi, Dr. Mansour Zand, and Dr. Zhenyuan Wang, for taking the time to review my work and provide much needed critique to help improve my dissertation.

Table of Contents

List of Figures

List of Tables

1. Introduction

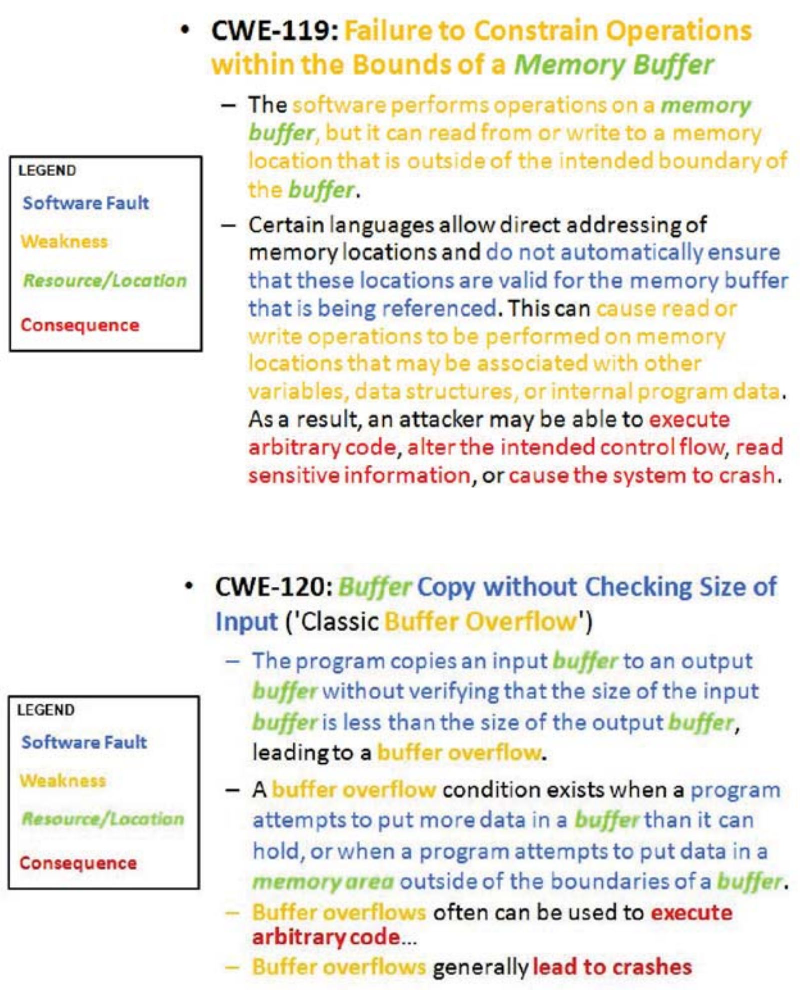
1.1 Problem Statement and Background

Software development is a highly error-prone process, which introduces a significant number of weaknesses into a system. Weakness is defined as a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software [42]. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC. Weaknesses in software products present opportunities for malicious attacks that compromise the integrity of the software, availability of its functionality and security of its data, due to validation errors, domain errors, and serialization/aliasing errors [24]. Vulnerability is an occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness [44]. The two important concepts are related but different. Weakness is a static presence that existing in software systems. Weakness might stay in software and never cause any problems until it is exploited by an attacker, and when the attacker find out the weakness (es) and exploit it (them), the vulnerability of this software is exposed. Most software development projects dedicate some effort to document, track and study reported vulnerabilities. This information is recorded in existing project repositories such as change logs 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 useful vulnerability-related information. In large projects, these repositories store vast amounts of data, and as a result, the relevant information is buried in a mass of other data. Natural language text 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 classifying and 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 in the face of 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. The large volume of data in software repositories and other project information sources make it difficult to locate the artifacts needed to identify, track and study previously recorded vulnerabilities. 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. Even if the information is found, a significant amount of work is needed to reconstruct the trail of artifacts that help understand the actual vulnerability. Thus, the information within software project repositories is not in a representation that can be easily extracted and analyzed for vulnerability-related questions.

We propose organizing the information in project repositories around semantic templates. We define semantic templates to be generalized patterns of relationship between software elements and faults, and their association with known higher level phenomena in the security domain. Semantic templates enhance the existing software project repositories by keeping track of relevant details and relating the information back to public vulnerability databases. In this approach, semantic templates for major vulnerability types are abstracted from information in the Common Weakness Enumeration dictionary [8]. Next, known vulnerabilities, related concepts in the 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.

The Common Weakness Enumeration (CWE) is a community developed dictionary of software weakness types. However, the CWE is large, abstract, contains multiple views and some imprecise and overlapping descriptions. This presents a huge cognitive overload for understanding a weakness, its preceding software faults, resources/locations that it occurs in, and following consequences. As shown in Figure 1, even the natural language descriptions contain tangled concerns representing different aspects of a single vulnerability. These concerns are also repeated in closely related CWE entries (examine the overlapping concerns in Figure 1).



- **CWE-119:** Failure to Constrain 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***.
  - Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data. As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

LEGEND
Software Fault
Weakness
Resource/Location
Consequence

- **CWE-120:** *Buffer* Copy without Checking Size of Input ('Classic Buffer Overflow')
  - The program copies an input ***buffer*** to an output ***buffer*** without verifying that the size of the input ***buffer*** is less than the size of the output ***buffer***, leading to a buffer overflow.
  - A buffer overflow condition exists when a program attempts to put more data in a ***buffer*** than it can hold, or when a program attempts to put data in a ***memory area*** outside of the boundaries of a ***buffer***.
  - Buffer overflows often can be used to **execute arbitrary code**...
  - Buffer overflows generally **lead to crashes**

LEGEND
Software Fault
Weakness
Resource/Location
Consequence

Figure 1 - Identification of Orthogonal Conceptual Units tangled in Natural Language CWE description

Closely related to the CWE, Common Vulnerability and Exposures (CVE) [40] is collecting known vulnerabilities from software development organizations, coordination centers and individuals, facilitating review by experts and assigning a unique identifier to each. Efforts exist to link the most recent CVE entries to corresponding CWE entries, but the links are weak and scarce. To build a richer context around existing vulnerability information, compensate for the missing information in vulnerability databases, and facilitate linking to known weakness classifications, we mine software repositories from open source projects. Software repositories also provide information about fixes, who discovered the vulnerability, who fixed it, etc., which compliment current vulnerability knowledge sources.

1.2 Research Hypotheses

This research is guided by the following study questions and corresponding hypotheses, which will be investigated.

1. How can semantic templates improve the ability of stakeholders in the software development life cycle (SDLC) to study software vulnerabilities?
    a. Semantic templates improve the ability to understand and discover vulnerabilities from scattered sources.
        i. Semantic templates improve the annotation and integration of vulnerability information in scattered sources.
        ii. Semantic templates improve the presentation and navigation of related vulnerability information.
    b. Semantic templates improve the ability to evaluate possible fixes to an occurrence of vulnerability.
        i. Semantic templates enable the categorization and cataloguing of past fixes to vulnerabilities.
        ii. Semantic templates help retrieve fixes from closely related vulnerabilities.
    c. Semantic templates improve the measurement of secure coding practices.
        i. Semantic templates facilitate development of metrics for

assessing project and/or developer capability in producing secure code.

2. Does the creation of semantic templates improve the quality of existing standards and enumerations describing vulnerability content?
    a. Semantic templates improve the ability to review existing standards and enumerations.
    b. Semantic templates reduce inconsistent or redundant information in existing standards.

2. Theoretical Background/Literature Review

2.1 Organizing Software Repositories

Software project repositories provide rich insights into the development history of a software product. Examples include version control systems, Integrated Development Environment (IDEs), issue and bug tracking systems, and mailing lists [36]. Version control systems store the history of modifications done to the source code. In addition to storing the actual code changes, these also record who made the change, when it was made, what files were changed, and why the change was made. Version control data provide information about how an occurrence of vulnerability was fixed and the related social network information. Issue and bug tracking systems record requests to fix a bug or enhance some functionality. The information typically includes who submitted the bug or change request, when it was submitted and resolved, the resolution selected. Mailing list between developers and stakeholders is another rich source of vulnerabilities. In emails, the developers discuss how the bugs were found, the technique to fix them, the aftermath of fixation, the plans and so on.

To make sense of this large quantity of data stored within such systems, many efforts for organizing software repositories have been suggested. The semantics of repository data is determined by the diverse and complex interrelationships between various software entities, e.g., classes, functions, files, bugs, versions, etc. Hence, most proposals provide different ways of combining these pieces of data to identify related bug data or related change data, connect bug data to change data, connect email data to change data, and so on. The augmented information is then used to provide useful inferences such as bug prediction, change impact assessment, identification of co-changed entities, identification of developer expertise, change effort estimation, etc. [36]. Given the usefulness of augmenting the raw repository data with these semantic relationships, it makes sense to store this augmented information in some form of enhanced repository. Examples of such systems are Kenyon [4] and TA-RE [22]. These systems ease the fact extraction process by providing a repository that combines multiple sources of information, particularly from version control data and information extracted from parsing individual source code versions, but for software evolution purpose and focus on the configuration aspect, not targeting at vulnerabilities.

2.2 Applying Ontologies in Software Engineering

Many research efforts have focused on modeling software repository data using ontologies. Ontology is a formal explicit specification of a shared conceptualization [13]. They provide a way of organizing and encoding the collected knowledge for a given domain. Ontology languages such as the Web Ontology Language (OWL) [45] enable the description of relationships that are constrained via description logic axioms, making it possible to formally qualify when two entries are related. Most ontologies also provide built-in inference tools for ease of querying for related information. One such example is EvoOnt [21], an OWL-based ontology that includes representations for software entities (e.g., classes, functions, variables, etc.), version control data and bug information. We also described how to utilize ontology to assess software engineering experiment designs [31] based on modeling and reasoning mechanisms provided by the ontology editor Protégé [12]. Each of these ontologies provides an infrastructure that makes it possible to execute higher level queries by reducing the incidental and effort-intensive task of data extraction. In concept, the proposed semantic templates take these approaches one step further by narrowing the possible queries under consideration down to one domain (security) and augmenting the information with higher level, domain-specific data.

2.3 Information Extraction and Semantic Annotation

Information Extraction deals with identifying and extracting a subsequence from a given sequence of instances that represents information we are interested in. The extracted data are annotated with specific information on the basis of some pre-defined metadata. Three state-of-the-art methods for information extraction are introduced in [30]: rule learning, classification and sequential labeling.

Content annotation for semantic web is also related to this research. Most research in this area focuses on semantic web construction. [28] introduces a methodology to partially annotate text content in web resources by an automatic and unsupervised way. The authors discover the relevant entities in text and associate them to classes of a seed ontology by using several learning techniques and heuristics, which inspire methods for content annotation using semantic structures but applied in a different domain. Our research is focused on the vulnerability domain which is more specific and need more expert guidance compared with the semantic web's general content. Similar research was conducted by D. Embley et al. [9]. They focused on converting regular web pages into semantic web pages so that ordinary users can utilize the free-form, textual queries to locate information they need with better efficiency and accuracy. It is also one of our objectives to improve the query mechanisms for developers to easily find relevant vulnerability information.

2.4 Text Mining

Text mining is a subfield of data mining. Data mining is an interdisciplinary field with its roots in databases, machine learning and statistics [16]. Generally, data mining has two major goals: prediction and description. Prediction aims to predict unknown or future values of other interest to using existed data set. Description is to uncover the hidden patterns within the data that can be useful to humans. The techniques to achieve these two goals are as follows [14][20]:

1. Classification: a process to discover the functions that classifies data objects into predefined classes.
2. Clustering: a process to identify a limited set of groups or clusters to describe the data.
3. Regression: a process to discover the functions that describe the relationships between data objects.
4. Summarization: a process to discover methods to describe the specific data set compactly and knowledgeably.
5. Dependency Modeling: a process to discover models that describe important dependencies between data objects in the data sets.
6. Change and Deviation Detection: processes to identify the significant changes in the data set.

As a branch of data mining, text mining discovers knowledge in unstructured data sets, such as text. This concept is first mentioned by R. Feldman et al. [10]. It exploits the techniques from information retrieval, information extraction and natural language processing (NLP) and connects them by the algorithms and methods of data mining. Our previous work [37] utilized clustering and classification techniques in text mining to group software engineering research papers. We apply similar text mining techniques in classifying vulnerabilities based on their natural language descriptions.

2.5 Vulnerability Classification and Categorization

Safety and security engineers can learn a lot from past incidents. However, given the volume and rapid rate at which new security 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] [5]. The software flaw taxonomy [24] 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. This classification being primarily top-down, details about a specific technology, platform or implementation language was to be inferred from the examples of vulnerabilities classified under each concept. Over a period of time several other refinements and ways to classify vulnerabilities were introduced [6] [2] [37]. More recently vulnerability categorizations have been developed as enumerations of weaknesses [7], ranked lists based on frequency of occurrence [43], domain-specific lists [44], or based on experience of security practitioners [17] [36].

To consolidate these efforts, The Common Weaknesses Enumeration is a community driven and continuously evolving taxonomy of software weaknesses that range from abstract categories to technology and language dependent categories. The CWE vision is to enable a more effective discussion, description, selection, and use of software security tools and services that can find weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design [8]. However, 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, as shown in Figure 2. While it provides a comprehensive record of software weaknesses and their categorization, it can be a daunting task for stakeholders in the SDLC to untangle and trace the complex web of interdependencies that exist among software weaknesses captured in the CWE. The expression of weakness classes and categories in the CWE often mix the characteristics of a weakness, its preceding software faults, resources/locations that the weakness occurs in, and the consequences that may follow from the weakness.

Closely related to the CWE, the Common Vulnerability Enumeration (CVE) is a growing compilation of known information security vulnerabilities and exposures as reported by software development organizations, coordination centers, developers and individuals at large. It provides a common standard identifier for each discovered vulnerability to enable data exchange between security products and provide a baseline for evaluating coverage of tools and services [40]. With the CWE coming into existence, new CVE entries are manually tagged with corresponding CWE concepts in National Vulnerability Database (NVD) [http://nvd.nist.gov].

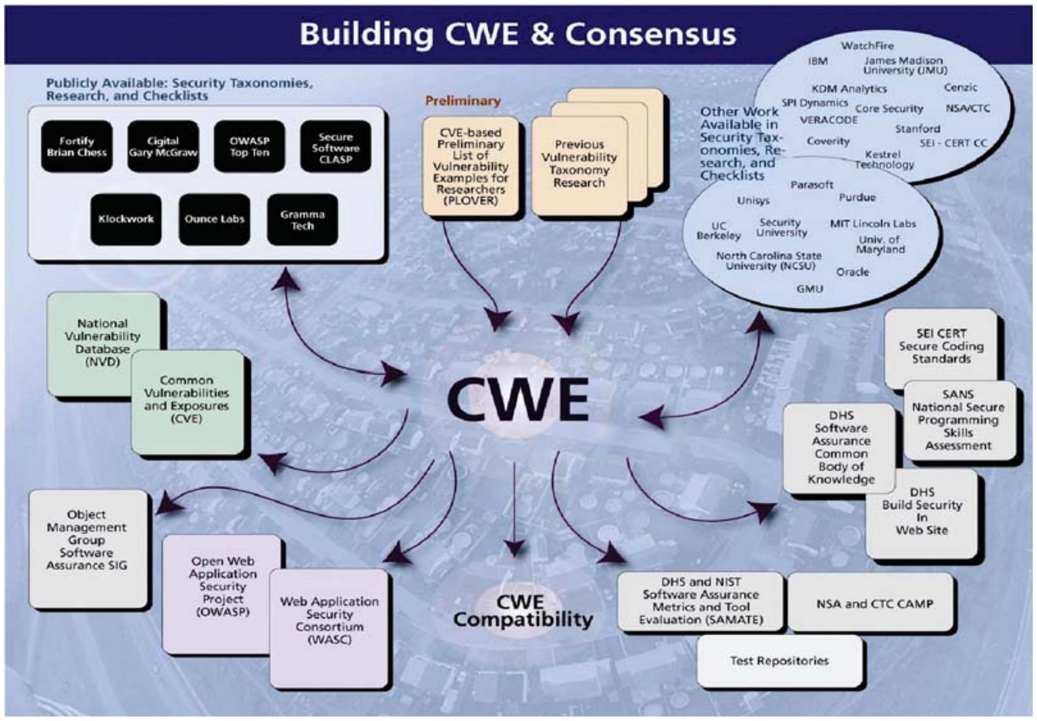

Building CWE & Consensus
Publicly Available: Security Taxonomies, Research, and Checklists
Fortify Brian Chess
Cigital Gary McGraw
OWASP Top Ten
Secure Software CLASP
Klockwork
Ounce Labs
Gramma Tech
Preliminary
CVE-based Preliminary List of Vulnerability Examples for Researchers (PLOVER)
Previous Vulnerability 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 Berkeley
Security University
MIT Lincoln Labs
Univ. of Maryland
North Carolina State University (NCSU)
Oracle
GMU
National Vulnerability Database (NVD)
Common Vulnerabilities and Exposures (CVE)
CWE
SEI CERT Secure Coding Standards
SANS National Secure Programming Skills Assessment
DHS Software Assurance Common Body of Knowledge
DHS Build Security In Web Site
Object Management Group Software Assurance SIG
Open Web Application Security Project (OWASP)
Web Application Security Consortium (WASC)
CWE Compatibility
DHS and NIST Software Assurance Metrics and Tool Evaluation (SAMATE)
NSA and CTC CAMP
Test Repositories

Figure 2 - CWE collaborations and contributions [http://cwe.mitre.org/]

2.6 Bug Fix Patterns

Identifying relevant fixes to vulnerabilities from a catalogue of reviewed and tested fix patterns provide a reliable guide for fixing a newly reported vulnerability with minimal side effects. To systematically identify relevant fixes, it is necessary to precisely understand the vulnerability at hand and its root causes. A fixed vulnerability reported in the change history does not mean the mechanism behind the vulnerability was really understood and the relevant part of the source code that led to the vulnerability was located. In [27], bug fix patterns in multiple Java projects have been catalogued examining only source code changes but not higher level information collected from different repositories and the weakness databases. The same group of authors also worked on helping locate bugs with the help of a bug finding algorithm based on the project-specific bug and fix knowledge base developed by analyzing the history of bug fixes [23]. The authors discovered certain fix patterns but just utilize them on finding bugs in specific projects, and also, this piece of work is still working on the low level source code only, without higher level knowledge adopted. Sudhakrishnan, et al. [32] provided a category of 25 bug fix patterns by analyzing the bug fix history of four hardware projects written in Verilog, which is an interesting work although based on certain hardware programming language and also in a source code level.

2.7 Timeline Analysis and Social Network Analysis

Every project has different technology, business objectives, organizational policies, processes and culture, and developer groups. Software repositories provide a richer temporal and social context for prioritization, measurement and attribution of vulnerability sources. There are several lines of investigation dealing with detecting temporal trends. The research described in [30] performs trend analysis in order to find hot topics through controlled vocabulary terms based on the nature of news that smaller units could be used to identify breaking news topics within short period such as one day. Temporal Text Mining (TTM) described in [26] is used to discover and summarize the evolutionary patterns of themes in a text stream over time. Based on the discoveries in characteristic path, authors of [15] collected the paper titles from

DBLP XML files to track the most popular terms used throughout time. Then they listed the emerging popular terms for each year by deleting terms that appeared in the previous two years and by this way they explained the previous discoveries.

Social Network Analysis in software projects helps identify relationships between developers and can provide better coordination between development teams and individuals, which was clearly validated by the survey conducted in [3]. The authors developed a tool named Codebook, which discovers transitive relationships between people, code, bugs, test cases, specifications, and other related artifacts by mining software repositories. It extensively supports multiple information needs with one data structure (a directed graph) and one algorithm.

3. Objectives of the Research Work

The central objectives of this dissertation research work include:

- Integrating vulnerability related information from large software repositories and the vulnerability research community, then organize them in such a way that semantic relationships between various elements are recorded;
- Applying this information to identify, semi-automatically annotate and discover vulnerabilities in existing and future software systems;
- Conducting an empirical study by using our approach to study vulnerabilities recorded in large open source software repositories and validating the effectiveness and efficiency of this approach.

This research work is distinguished from the previous research in the following aspects:

1. We use a top-down (semantic templates) and bottom-up (repository mining) approach to organize vulnerability related concepts, properties, relationships and instances instead of a loose list of related information without real structure [7]. (maps to sections 2.1, 2.3, 2.5)
2. Rather than generic and often superficial tools, we focus on a single domain related to security vulnerabilities for deeper, more meaningful guidance in identifying and fixing vulnerabilities. (maps to sections 2.1, 2.2, 2.6, 2.7)
3. We expect to improve the developer ability to identify effective project specific fixes by examining past vulnerability information. This guidance relies on semantic relationships in addition to specific code level descriptions of possible

fixes. (maps to section 2.6). We expect to make the existing weakness enumerations more usable during the software development lifecycle and project management. (maps to section 2.5)

4. The vulnerability related information considered for this research is collected from structured as well as unstructured sources in software repositories. Most existing works in vulnerability mining from software repositories rely on one or two information sources such as source code [4] and bug reports. (maps to sections 2.1, 2.2, 2.3, 2.4)
5. Timeline-based vulnerability trend analysis, visualization and social network analysis will provide unique insights to understand and discover novel aspects of vulnerabilities. (maps to section 2.7)

4. Research Methodology

The research work performed for this dissertation can be divided into three parts. First is the development of a semantic infrastructure for vulnerability analysis. For each category of vulnerabilities, a semantic template was constructed to organize the four most important concepts: Software fault, Weakness, Resource/Location and Consequence. This includes a project-specific set of studies of vulnerabilities carried out on the Apache Web Server repositories, which provides an illustration of how the application of the semantic infrastructure facilitates and enables more sophisticated analysis of the available repository information. Second is an empirical validation to assess how well semantic templates enable the comprehension of the vulnerability information retrieved from software repositories. Finally, we investigate a machine learning approach, for semi-automatic annotation of CVE descriptions using concepts in the semantic templates.

4.1 Semantic Infrastructure for Vulnerability Analysis

In this stage there are five sets of activities described as following:

- Analysis of known vulnerabilities in Apache Web Server project to identify most common vulnerability types.
- Constructing semantic templates for different types of vulnerabilities to show the knowledge structure in vulnerability area as an organized structure from both the Weakness research community and the software repository knowledge.

- Use semantic templates to annotate vulnerability instances from Apache Web Server project, and further realize the semi-automatic process to help annotation works for human experts;
- Ontology Construction to help represent and maintain the concepts, relationships and properties contained in the vulnerability-related semantic templates;
- Visualization of the vulnerability concepts structure. GraphViz is used to represent the concepts as nodes and relationships as arcs to provide a high level structure in certain category.

4.1.1 Analysis of Project Vulnerabilities

The purpose of trend analysis is to identify the distribution of different categories of vulnerabilities, software faults, weaknesses, resources, consequences and their trends with respect to time. This analysis guides the selection, prioritization, evaluation, and adjustment of granularity of concepts in semantic templates. The methodology as outlined in Chapter 5 to construct semantic templates covers the most common types of vulnerabilities in the Apache Web Server occurring over time.

4.1.2 Development of Semantic Templates

We define semantic templates to be generalized patterns of relationship between software elements and faults, and their association with known higher level phenomena in the security domain. Semantic templates enhance the existing software project repositories by keeping track of relevant details and relating the information back to public vulnerability databases. In this approach, semantic templates for major vulnerability types are abstracted from information in the CWE. Next, known vulnerabilities, related concepts in the 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.

Semantic templates provide a meaningful context for understanding entire categories of vulnerabilities. With the help of software repository information such as log of changes, code differences before and after the fix, the vulnerabilities that occurred in Apache Web Server project can be analyzed with much more understanding. Annotate the vulnerability instances in Apache Web Server project by the corresponding semantic template to help better understanding.

4.1.3 Ontology Construction and Querying

The ontology construction process identifies concepts and relationships which between concepts, properties and instances in a given domain based on certain purpose, and then integrates them into a meaningful structure that facilitates further inferences. In this research, the domain is based on categories of vulnerabilities in Apache Web Server project. Protégé is used as the ontology editor because of its powerful and extensive functionalities for representing and inferring knowledge. The most important resource included in this ontology construction process is the corresponding semantic template. The nodes in the semantic template are treated as concepts in ontology, and the relationships between nodes are modeled as relationships in ontology. Additional information includes CWE entry names and descriptions, which can hint at properties and constraints in the ontology. The ontology as constructed provides a framework to describe the knowledge structure in this domain, and to understand specific instances of vulnerabilities.

The descriptive logic based reasoning capability offered by ontologies facilitates further analysis activities of related vulnerability data. This makes it possible to ask, for example, if a resource was involved in multiple vulnerability occurrences. Furthermore, the combination of ontological information and data from project repositories can enable the discovery of previously undetected or unreported vulnerabilities, e.g., in related components that share a vulnerable resource. Additional queries can be developed to provide preliminary answers to hypotheses concerning causes leading to vulnerabilities, relationships between faults and vulnerabilities, or relationships between vulnerability types, etc. Analysis of occurrences of individual concepts in the semantic template with respect to time can also be formulated.

4.1.4 Interactive Visualization

Interactive visualization of large amounts of information provides one way of dealing with the information overload. As most of the information here is relational in nature, they lend themselves to graph or network visualization tools which can be used to depict direct and indirect relationships between different nodes. Most of such tools provide the capability to layout the graph automatically, pan and zoom the graph, highlight sets of interesting nodes, filter out uninteresting ones and dynamically adjust the layouts. An approach for visualizing the CWE hierarchy using GraphViz has been developed and further discussed in chapter 5. This visualization will continue to be modified and refined to assist in the comprehension of existing structures. GraphViz is limited in that it does not provide a way to dynamically filter or change the layout once it is rendered. Other graph visualization tools will also be explored to investigate their dynamic capabilities.

4.2 Empirical Validation

An experiment on software engineering students is carried out to assess the feasibility and effectiveness of semantic templates for reviewing and understanding vulnerabilities. Participants are asked to study vulnerabilities reported by the Apache Web Server project and answer a set of questions afterwards. The students will be randomly assigned to two groups, one group is given semantic templates and the other group is not. The time spent and the accuracy of answers for the two groups is used to evaluate the effectiveness of semantic templates in understanding certain classes of vulnerabilities.

4.3 Tool-Supported Classification and Annotation of Repository Entries

Annotation refers to the labeling or mapping of repository and vulnerability information elements to the appropriate concept elements in the semantic template. Previously, this annotation process was manually performed. While manual annotation is useful for recording new vulnerabilities as they are detected, to relate a large number of past vulnerabilities with the templates requires automation. The appropriate semantic template, and the most fitting concepts within that template, needs to be identified. Using a machine learning approach, we develop classifiers to accomplish both tasks: a high level weakness category classifier for selecting the most likely weakness category and its corresponding semantic template (for example, is it a buffer overflow or injection?), and a concept classifier for selecting the most likely concepts within that semantic template. The classification results can then be used to annotate vulnerabilities with corresponding security concepts. As a by-product, previously unreported vulnerabilities may also be identified.

In this machine learning approach, weakness category classification consists of training a classifier to recognize vulnerability reports whose categories are known. Concept classification uses CWE entries as a training set to classify version repository logs and CVE descriptions into applicable semantic template concepts. As will be shown in Chapter 7, natural language descriptions, source code fixes are mapped to elements in specific categories (Software Fault, Weakness, Resource, and Consequence). In effect, the annotation process will populate the semantic templates with instance information gleaned from repository and vulnerability information. Text mining techniques will be employed to analyze textual information in version logs and CVE and CAPEC entries to identify relevant information elements.

It is expected that mapping a collection of CVE vulnerabilities to specific weakness categories will reveal recurring error patterns and provide project specific measures for identifying the most prominent weaknesses for which developers need awareness and training. Mapping of the semantic template to CAPECs helps develop and prioritize test cases for the most exploited software flaws revealed from past CVEs.

5. Semantic Template Construction

We are faced with two problems in vulnerability research works: information overload in weakness enumerations and categorization; at the same time, paradoxically, lack of information or security know-how among SDLC (Software Development Life Cycle) stakeholders to avoid the weakness. For the former problem we outline a stepwise and repeatable process [11] to extract a set of weaknesses, from abstract to more specific, from the CWE pertaining to a class of weakness. For the latter problem, we systematically “isolate” the conceptual units previously combined in a CWE entry and explicate the relationships among these conceptual units to form a coherent semantic template. Constructing the semantic templates for different types of vulnerabilities is an iterative and exploratory process. We first describe the Apache Web Server vulnerability analysis which drove the development of semantic templates, followed by a detailed description of the construction process itself.

5.1 Analysis of Apache Web Server Vulnerabilities

Before semantic templates are constructed, Apache Web Server vulnerabilities are analyzed to identify recurring vulnerabilities, which will be used to select, prioritize, evaluate, and adjust granularity of concepts in semantic templates.

We performed a timeline analysis of reported vulnerabilities, which is based on the manual classification of Apache Web Server vulnerabilities by different experts and statistical processing over time, with the period of every year from 1998 to 2010. Since vulnerabilities recorded in the National Vulnerability Database (NVD) [http://nvd.nist.gov] are generally under review for a long time, we relied on the dates in the change history and mailing lists to ascribe a time stamp to each occurrence of vulnerability. The results are shown in Figure 3. The categories identified are:

- Injection
- Injection in the form of cross-site scripting (inj/xss), which forms a large subclass of injection
- Information exposure
- Denial of service (dos)
- Resource exhaustion leading to denial of service (dos/resource)
- Buffer overflow

We can draw several observations here. For example, as shown in Figure 3, it is clear that exposure, denial of service and injection are recurring vulnerabilities which almost span every year, but buffer overflow vulnerabilities occurred mostly from 2002-2005. Given more information, it would be interesting to uncover what happened in this period that led to a large number of Buffer Overflow vulnerabilities. With the same process, annotated Apache Web Server vulnerabilities can be analyzed by other dimensions as software faults, weaknesses, resources and consequences so that the distribution and trend on those dimensions can be examined, e.g. during a certain period, heap-based buffer overflow may present very intensively so that indicating a period with more usage of heap-based buffer in programming.

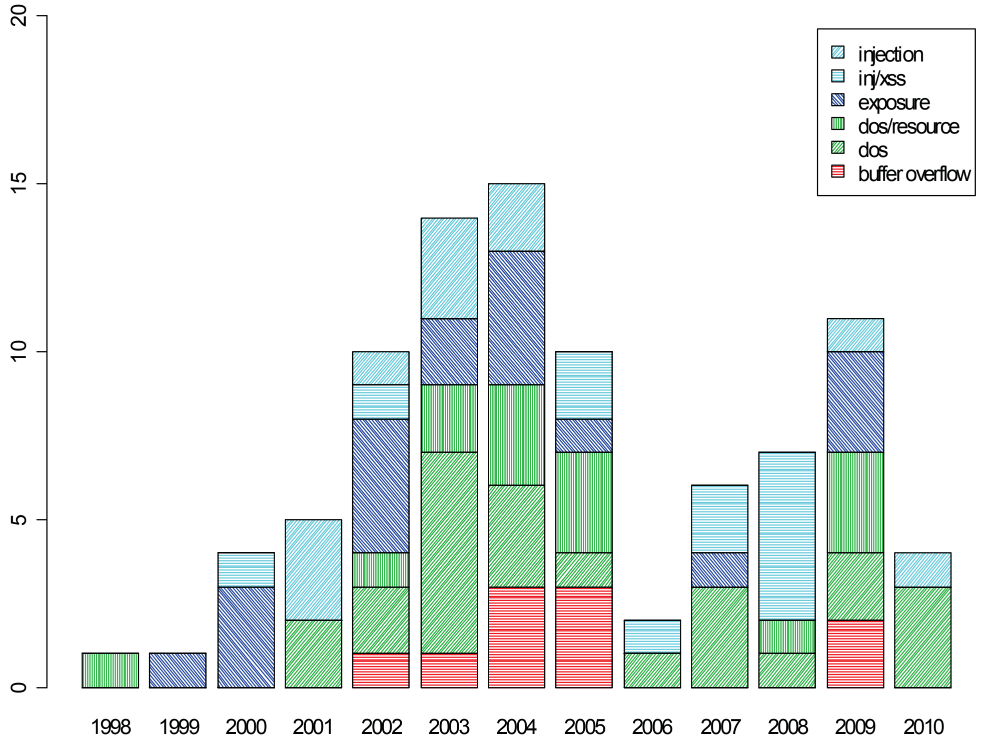


Figure 3 - Occurrences of various vulnerability categories over time

Based on Figure 3, if we subsume cross-site scripting and resource exhaustion under their respective classes, the most commonly occurring vulnerabilities are injection, information exposure, and denial of service and buffer overflow.

5.2 Process for Building a Semantic Template

This process began with selecting a proper vulnerability category. Of the four categories identified in the trend analysis, we started with Buffer Overflow. The reasons for choosing Buffer Overflow vulnerabilities include that first, buffer management is a usual part that developers can easily make mistakes and second, Buffer Overflow vulnerabilities are most widely occurring and have been around for much longer than the others. So Buffer Overflow was chosen as the example in this dissertation. The second chosen vulnerability category is injection, which is receiving a lot of attention more recently. With the experiences from constructing the buffer overflow semantic template, the process to produce semantic template for injection vulnerabilities is more complete and efficient compared with others. Finally, information exposure was the third constructed semantic template.

For each major class of weakness, such as “buffer overflow” or “injection” in the Apache Web Server a large number of CWE entries can be identified. In addition, a significant amount of work is needed to reconstruct the trail of CWE entries such that the chain of events that lead to a vulnerability listed in the CVE can be reconstructed. The conceptual units contained in the semantic template are extracted by asking these simple but important security questions for each CWE entry extracted for a class of weakness: 1) What software faults, i.e. concrete manifestations of flaws in the software program and design related to omission, commission or operational, can precede the weakness? 2) What is the core defining characteristic of the weakness? 3) What are the resources and locations where the weakness commonly occurs in? 4) What consequences, i.e. failure conditions that violate security properties, the weakness can precede? The creation of a semantic template can be viewed as a process of untangling the relevant information to certain groups of CWE entries into different aspects that can be later weaved on-demand to explain phenomena behind an occurrence of vulnerability.

We now describe the semantic template construction process. Here the preparation and collection phase of this approach is described.

A. Preparation and collection phase:

A.1 Selection of content:

The CWE is a continuously evolving effort. Distinct milestones in the CWE are assigned versions. Therefore, the first task is identifying a baseline CWE version to be used for the semantic template. The current effort of this research is based on CWE v1.6 specification [8]. The CWE specification used "views" to organize and examine its contents of entries, and then different slices of structure can be utilized in different environment. Based on the research purposes of this dissertation, the Research (CWE-1000) and Development (CWE-699) graph views were chosen to identify relationships between CWE entries.

A.2 Extraction of relevant weaknesses:

The next step is to identify the specific CWE entry that indicates the characteristics of weakness of interest at the most abstract level. For the "Buffer Overflow" weakness category, CWE – 119 "Failure to Constrain Operations within the Bounds of a Memory Buffer" is such an entry. It is called the root entry for this specific vulnerability category. Starting with the root entry, four strategies were adopted to gather weaknesses related to it in both of the CWE research and development views. The four strategies to identify related weakness entries are:

a. Navigating hierarchical structure of the root entry (specializations and generalizations) by the ParentOf and ChildOf relationships.

b. Navigating non-taxonomical relationships such as “Can Precede”, “Can Follow”, “Peer-of” in the CWE hyperlinked document [8].

c. Keyword search on the CWE hyperlinked document [8] for weaknesses that include the important keywords in the primary or extended description, e.g. “overflow” or “off-by-one” for Buffer Overflow vulnerability category. Keyword search is followed by exploration of parent, sibling and child categories of the discovered CWE, for relevance to the root entry.

d. Visualization of the CWE XML specification [8] by using Graphviz [18] visualization utilities so that the whole picture could be grasped at the mean time of detailed researching. Figure 4 showed the Buffer Overflow related graph resulted by the four strategies.

The strategies a and b are basically executed automatically by running script analyzer on the CWE hyperlinked document so that all the CWE concepts having a path linked with the root entry directly or indirectly are identified. While applying each strategy, use of heuristics and some degree of judgment is required on part of the subject matter expert to include a CWE entry into the pool of relevant entries. For example, if keyword search surfaces a CWE entry that belong to a different category then only that entry and its parent category are added to the relevant set. In other cases the complete trail between the discovered CWE and the root entry may be added.

To mitigate subjective opinions as much as possible in selection, multiple experts (in this case the author and two of the committee members) independently apply the strategies. Later the results from all experts are consolidated in a group workshop. The output of this phase is a set of weaknesses closely related to the root entry.

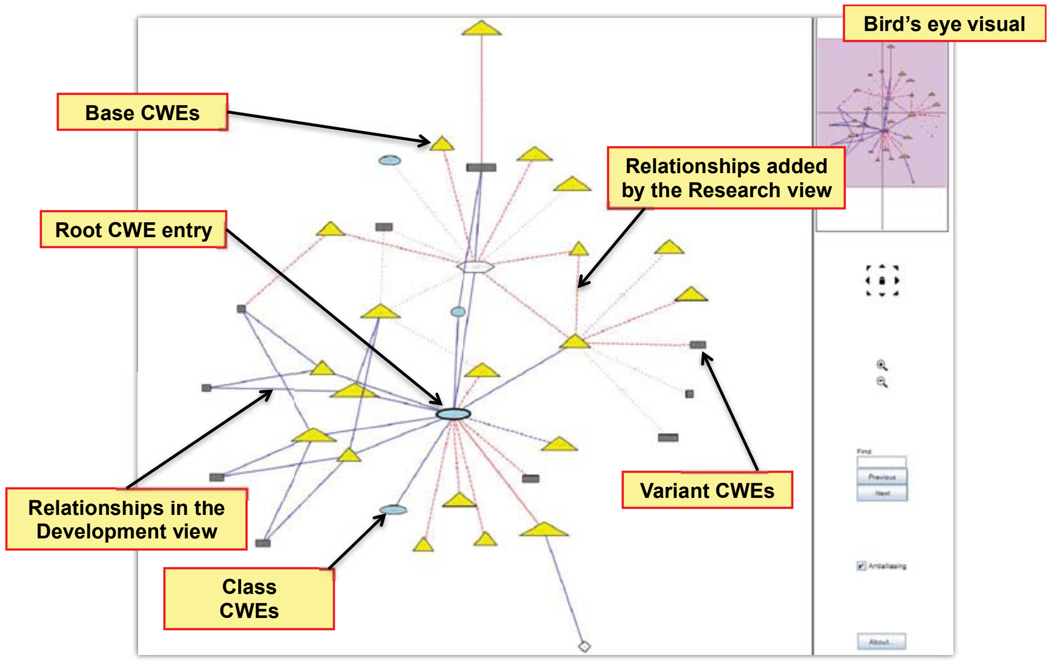

Bird's eye visual
Base CWEs
Relationships added by the Research view
Root CWE entry
Variant CWEs
Relationships in the Development view
Class CWEs

Figure 4 - Visualization of Buffer Overflow-related CWE entries

Figure 5 speaks volumes about the complexity of the "mental model" that developers need to be aware of to understand the possible faults and consequences of their design and coding decisions by organizing the CWE entries that have direct or indirect relationships with the root entry CWE-119. Although hyperlinked, navigating the CWE documentation and various graphical representations is a tedious and non-intuitive work. While different CWE views help to accommodate multiple perspectives, it adds an additional layer of complexity to understand certain vulnerability. The CWE is comprehensive; however the current nested structure and tangled contents are confusing for stakeholders in SDLC to understand.

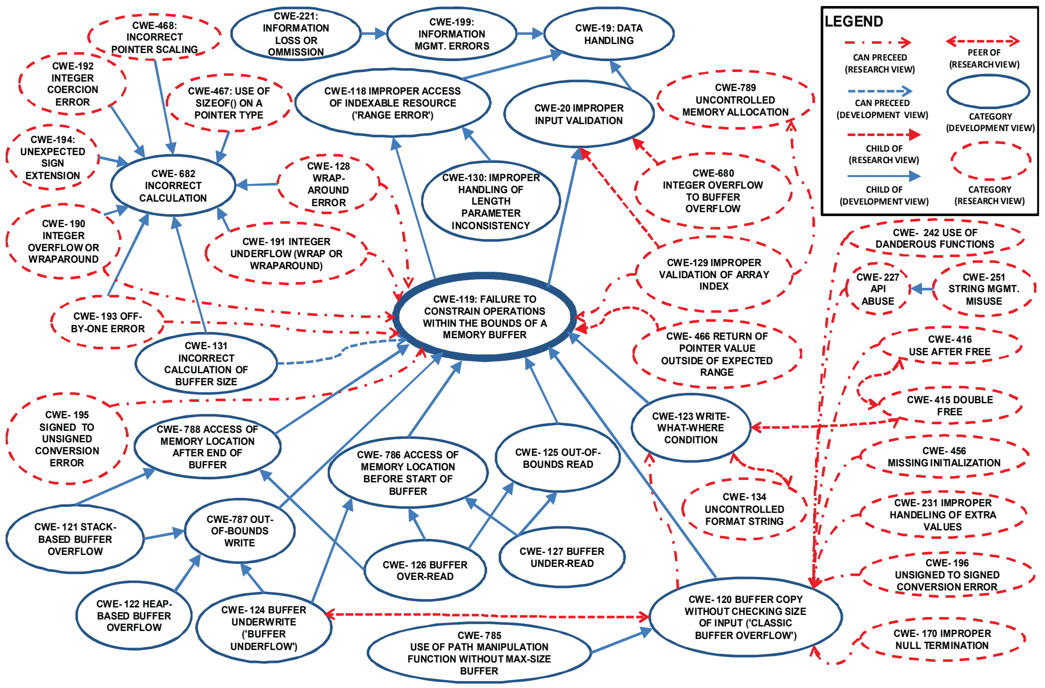

CWE-468: INCORRECT POINTER SCALING
CWE-221: INFORMATION LOSS OR OMMISSION
CWE-199: INFORMATION MGMT. ERRORS
CWE-19: DATA HANDLING
CWE-192 INTEGER COERCION ERROR
CWE-467: USE OF SIZEOF() ON A POINTER TYPE
CWE-118 IMPROPER ACCESS OF INDEXABLE RESOURCE ('RANGE ERROR')
CWE-20 IMPROPER INPUT VALIDATION
CWE-789 UNCONTROLLED MEMORY ALLOCATION
CWE-194: UNEXPECTED SIGN EXTENSION
CWE- 682 INCORRECT CALCULATION
CWE- 128 WRAP-AROUND ERROR
CWE-130: IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY
CWE-680 INTEGER OVERFLOW TO BUFFER OVERFLOW
CWE- 190 INTEGER OVERFLOW OR WRAPAROUND
CWE- 191 INTEGER UNDERFLOW (WRAP OR WRAPAROUND)
CWE-129 IMPROPER VALIDATION OF ARRAY INDEX
CWE- 193 OFF-BY-ONE ERROR
CWE-119: FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER
CWE- 131 INCORRECT CALCULATION OF BUFFER SIZE
CWE- 466 RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE
CWE- 195 SIGNED TO UNSIGNED CONVERSION ERROR
CWE- 788 ACCESS OF MEMORY LOCATION AFTER END OF BUFFER
CWE- 786 ACCESS OF MEMORY LOCATION BEFORE START OF BUFFER
CWE- 125 OUT-OF-BOUNDS READ
CWE-123 WRITE-WHAT-WHERE CONDITION
CWE- 134 UNCONTROLLED FORMAT STRING
CWE- 121 STACK-BASED BUFFER OVERFLOW
CWE-787 OUT-OF-BOUNDS WRITE
CWE- 126 BUFFER OVER-READ
CWE- 127 BUFFER UNDER-READ
CWE- 122 HEAP-BASED BUFFER OVERFLOW
CWE- 124 BUFFER UNDERWRITE ('BUFFER UNDERFLOW')
CWE- 785 USE OF PATH MANIPULATION FUNCTION WITHOUT MAX-SIZE BUFFER
CWE- 120 BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW')
LEGEND
CAN PRECEED (RESEARCH VIEW)
PEER OF (RESEARCH VIEW)
CAN PRECEED (DEVELOPMENT VIEW)
CATEGORY (DEVELOPMENT VIEW)
CHILD OF (RESEARCH VIEW)
CHILD OF (DEVELOPMENT VIEW)
CATEGORY (RESEARCH VIEW)
CWE- 242 USE OF DANDEROUS FUNCTIONS
CWE- 227 API ABUSE
CWE- 251 STRING MGMT. MISUSE
CWE- 416 USE AFTER FREE
CWE- 415 DOUBLE FREE
CWE- 456 MISSING INITIALIZATION
CWE- 231 IMPROPER HANDELING OF EXTRA VALUES
CWE- 196 UNSIGNED TO SIGNED CONVERSION ERROR
CWE- 170 IMPROPER NULL TERMINATION

Figure 5 - Buffer Overflow-related CWEs Extracted in the Preparation and Collection Phase

***B. Concept isolation and template structuring*:**

B.1 Separation of tangled conceptual units in CWE entries:

In this phase the set of CWE entries from the previous phase are carefully analyzed to identify distinct conceptual units as Software Faults, Weakness, Resource/Location and Consequences. For example, the conceptual units in CWE – 119 "Failure to Constrain Operations within the Bounds of a Memory Buffer" can be systematically identified as:

Software fault:

"Failure to Constrain Operations within the Bounds of a Memory Buffer"

Weakness:

"…read from or write to a memory location that is outside of the intended boundary…"

Resource/Location: "Memory buffer"

Consequences:

"…execute arbitrary code…", "…alter the intended control flow…", "…read sensitive information…", "…cause the system to crash…",

B.2 Filtering entries and introducing abstractions:

The CWE entries are specified as class, base or variant weakness, with class weakness being the most general. Class weaknesses are described in a very abstract fashion, typically independent of any specific language or technology. Base weakness is also described in an abstract fashion, but with sufficient details to infer specific methods for detection and prevention of themselves. Variants on the other hand are described at a very low level of detail, typically limited to a specific language or technology.

With the original intent of the semantic template to make weakness more understandable, the primary concepts of Software Fault and Weakness conceptual units in the template are derived from the more general class and base weaknesses, while preserving traceability to more specific variants using their CWE identifies. This design decision was primarily taken to avoid missing the forest for the trees. Hopefully it can make it easier for developers to remember a more generic model of the weakness rather than a detailed one.

However, it is not uncommon to extract concepts in the template from variant weaknesses in the case of Resources/Locations conceptual unit. For the Consequences conceptual unit, Observation was made that the concepts extracted from consequences listed for class and base weaknesses in the CWE provide comprehensive coverage of consequences identified from more specific weaknesses.

B.3 Template structuring and representation:

The concepts identified in each of the four conceptual units of the template are structured and related to each other based on the relationships between their corresponding CWE entries. From this effort a highly structured collection of interdependent Buffer Overflow concepts emerges as shown in Figure 7. Each concept in the semantic template of Figure 7 is traceable to corresponding CWE categories the first phase. The semantic templates are also represented in an OWL ontology format using Protégé [12]. The description logic based reasoning capability offered by Protégé facilitates further analysis activities of related weaknesses.

B.4 Template refinement and tailoring:

By mapping specific vulnerabilities (CVE [40]) to the semantic template, it is further refined and checked for obvious omissions. Such mapping in the context of the vulnerability CVE-2004-0492 in the Apache Web Server is described in the following section. It is also expected that the semantic templates can be tailored for a specific project, product or organization.

At the end of this detailed process, the Buffer Overflow semantic template was constructed as shown in Figure 6. The Buffer Overflow Semantic Template contains four groups of important concepts organized as software fault, weakness, resource/location and consequence between which “can precede” and “occurs in” relationship exist. The software fault contains groups of possible design or source code faults that may lead to buffer overflow. CWE-682 “Incorrect Calculation” is the root of the most condensed software fault tree, in which the infamous “off-by-one” error, “Integer Overflow”, “Sign Error” etc., as the more specific instances of incorrect calculation, are included as its children. Other than this incorrect calculation tree, there are several other possible software faults that can lead to buffer overflow, such as the “Improper Input Validation”, “API Abuse” and “Missing Initialization”. The presence of buffer overflow weaknesses is always “Improper Access of Index-able Resource” which represents the typical and abstract appearance of a buffer overflow in software. Resource component has a relatively simple structure with the “buffer” as the root resource which is part of the “indexable resource” and has child named as “memory buffer” and grandchildren “stack-based buffer”, “static buffer” and “heap-based buffer”. The most popular consequence of buffer overflow is “Write-what-where Condition” which means that the attacker has the ability to write an arbitrary value to an arbitrary location. The “Uncontrolled Memory Allocation” and “Information Loss or Omission” are also possible consequences of buffer overflow weakness.

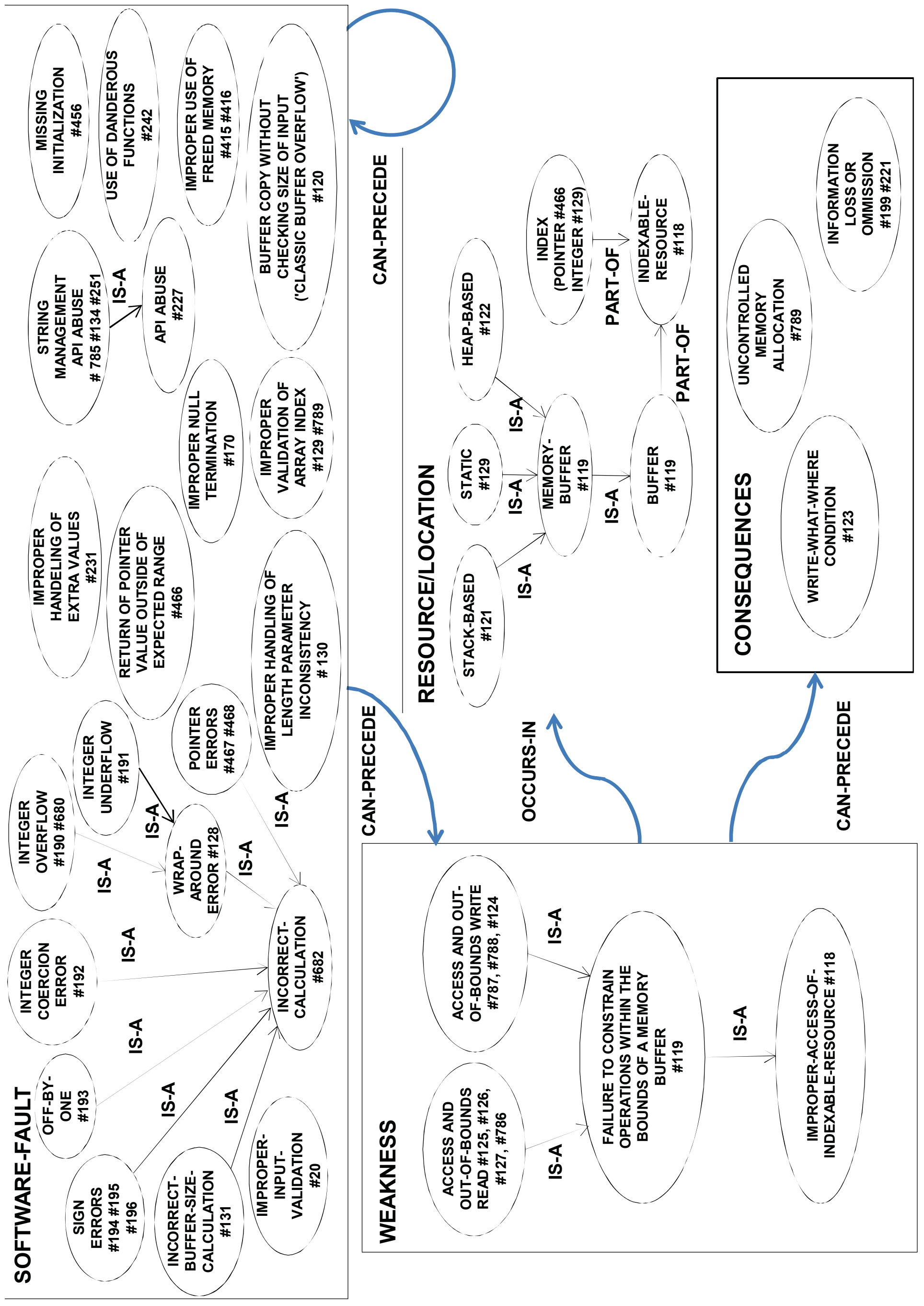

SOFTWARE-FAULT
MISSING INITIALIZATION #456
USE OF DANDEROUS FUNCTIONS #242
IMPROPER USE OF FREED MEMORY #415 #416
BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW') #120
STRING MANAGEMENT API ABUSE # 785 #134 #251
API ABUSE #227
IMPROPER NULL TERMINATION #170
IMPROPER VALIDATION OF ARRAY INDEX #129 #789
IMPROPER HANDELING OF EXTRA VALUES #231
RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE #466
IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY #130
INTEGER OVERFLOW #190 #680
INTEGER UNDERFLOW #191
POINTER ERRORS #467 #468
WRAP-AROUND ERROR #128
INTEGER COERCION ERROR #192
OFF-BY-ONE #193
SIGN ERRORS #194 #195 #196
INCORRECT-BUFFER-SIZE-CALCULATION #131
IMPROPER-INPUT-VALIDATION #20
INCORRECT-CALCULATION #682
IS-A
CAN-PRECEDE
RESOURCE/LOCATION
HEAP-BASED #122
INDEX (POINTER #466 INTEGER #129)
PART-OF
INDEXABLE-RESOURCE #118
STATIC #129
MEMORY-BUFFER #119
STACK-BASED #121
BUFFER #119
CONSEQUENCES
UNCONTROLLED MEMORY ALLOCATION #789
INFORMATION LOSS OR OMMISSION #199 #221
WRITE-WHAT-WHERE CONDITION #123
OCCURS-IN
WEAKNESS
ACCESS AND OUT-OF-BOUNDS WRITE #787, #788, #124
ACCESS AND OUT-OF-BOUNDS READ #125, #126, #127, #786
FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER #119
IMPROPER-ACCESS-OF-INDEXABLE-RESOURCE #118

Figure 6 - Buffer Overflow Semantic Template

5.3 Using the Semantic Template to Study Vulnerabilities

To validate the power of buffer overflow semantic templates to help understanding real world vulnerabilities, we manually annotated several instances of

Apache Web Server vulnerabilities from the buffer overflow category, by collecting the CVE descriptions from Apache Website, NVD entries, and the development repositories concerning specified vulnerabilities. By reviewing the existing vulnerabilities in the Apache Web Server project, several buffer overflow vulnerabilities were identified and the CAN-2004-0492 was chosen as the example for manual annotation which is shown in figure 7. The NVD description for CAN-2004-0492 mentioned: "…possibly execute arbitrary code via a negative Content-Length HTTP header field", the Apache Website description mentioned that "buffer overflow…can be triggered by receiving an invalid Content-Length header" and the fix of this vulnerability is to add a validation criteria to avoid a negative input, so for the software fault component, it is obvious that the fault belongs to CWE-20 "Improper Input Validation" and the CWE-130 "Improper Handling of Length Parameter Inconsistency". Similarly, the description in NVD about "allows remote attackers to cause a denial of service (process crash) and possibly execute arbitrary code" and "cause the Apache child processing the request to crash. This issue may lead to remote arbitrary code execution on some BSD platforms" in Apache Website tell us that the consequence of this vulnerability belongs to CWE-123 "Write-What-Where Condition".

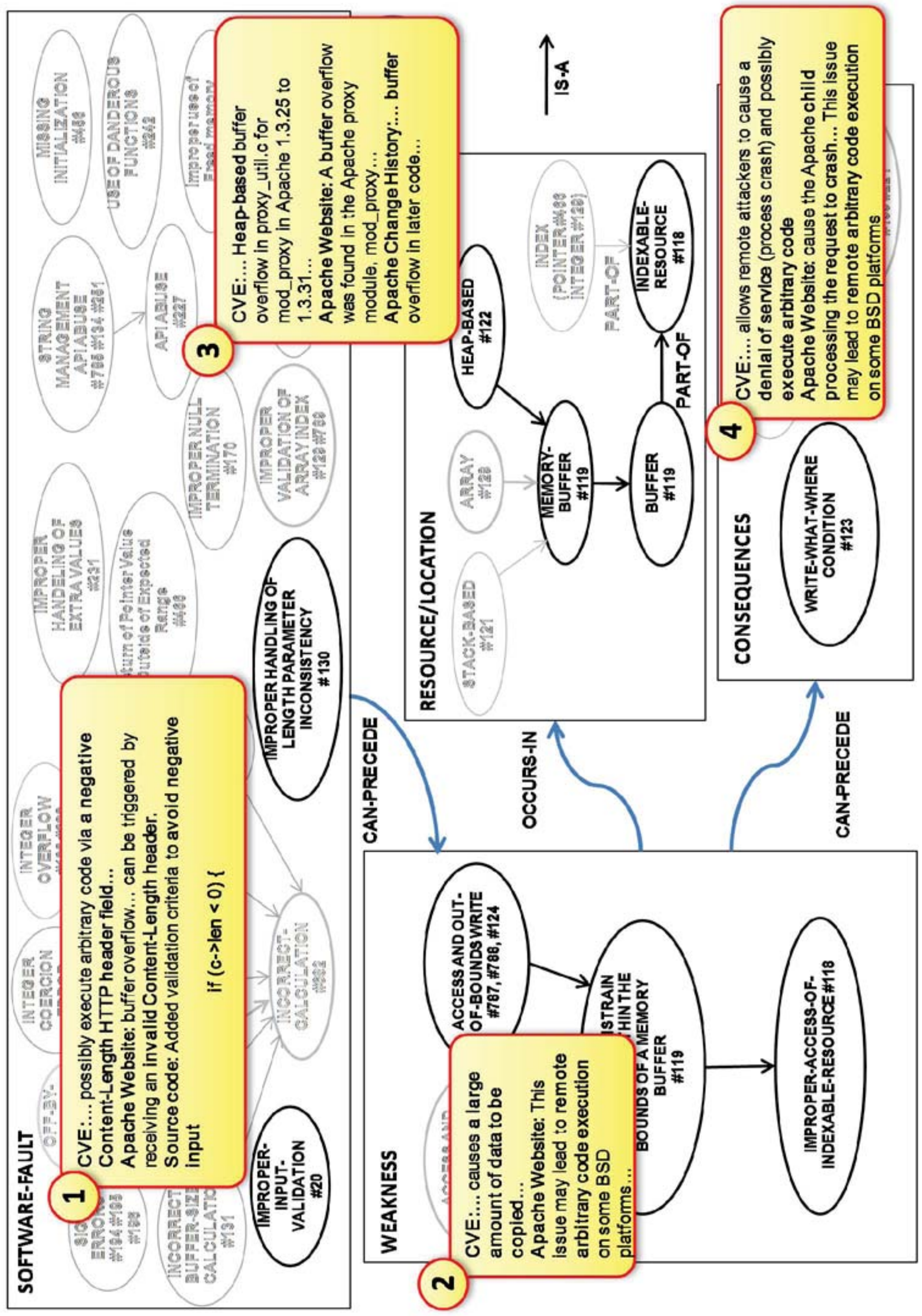

SOFTWARE-FAULT
IMPROPER-INPUT-VALIDATION #20
IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY #130
INCORRECT-CALCULATION #682
1 CVE:… possibly execute arbitrary code via a negative Content-Length HTTP header field… Apache Website: buffer overflow… can be triggered by receiving an invalid Content-Length header. Source code: Added validation criteria to avoid negative input
if (c->len < 0) {
3 CVE:… Heap-based buffer overflow in proxy_util.c for mod_proxy in Apache 1.3.25 to 1.3.31… Apache Website: A buffer overflow was found in the Apache proxy module, mod_proxy… Apache Change History:… buffer overflow in later code…
CAN-PRECEDE
OCCURS-IN
CAN-PRECEDE
RESOURCE/LOCATION
HEAP-BASED #122
STACK-BASED #121
ARRAY #128
MEMORY-BUFFER #119
BUFFER #119
INDEXABLE-RESOURCE #118
INDEX (POINTER #68 INTEGER #129)
PART-OF
PART-OF
IS-A
WEAKNESS
ACCESS AND OUT-OF-BOUNDS WRITE #787, #788, #124
…STRAIN …THIN THE BOUNDS OF A MEMORY BUFFER #119
IMPROPER-ACCESS-OF-INDEXABLE-RESOURCE #118
2 CVE:… causes a large amount of data to be copied… Apache Website: This issue may lead to remote arbitrary code execution on some BSD platforms…
CONSEQUENCES
WRITE-WHAT-WHERE CONDITION #123
4 CVE:… allows remote attackers to cause a denial of service (process crash) and possibly execute arbitrary code Apache Website: cause the Apache child processing the request to crash… This issue may lead to remote arbitrary code execution on some BSD platforms

Figure 7 - Annotated CVE-2004-0492 by Buffer Overflow Semantic Template

5.4 Other Semantic Templates

By following the same procedure indicated in previous section, the semantic template for Injection vulnerabilities is also constructed as shown in figure 8 and 9. Based on the Injection semantic template, the main group of software faults that can precede the injection weakness has the center fault named as “Failure to Sanitize User input of Syntax that has implications in a Different Plane”, which was extracted from the root entry CWE-74 and shared with CWE-74, CWE-94, CWE-99 and CWE-138. As the more specific software faults, the CWE-75 to CWE-91 and several other CWE entries were summarized into seven children software faults of this center fault, with the “is-a” links. In this tree-like software fault concept group, all of the concept nodes share the element of “failure to preserve/sanitize” certain structures. At the same time, one of the child node has its own children, and this center fault node “Failure to Sanitize User input of Syntax that has implications in a Different Plane” is the child of another two concepts “improper enforcement of message or data structure” and “improper input validation” (also is a software fault in buffer overflow semantic template as a general fault) with the “is-a” relationship. Comparatively the weakness concept structure is simple, with only one element “Elements of User-controlled Data Have Implications in a different plane”. The consequence panel is also relatively simple, with only one tree rooted in the element “Execution of Arbitrary User-Controlled Data”, with six children as the specific consequences. The most complicated part is the resource. With the root of “User-controlled input Data”, the injection weakness occurs in various resources at different levels and locations.

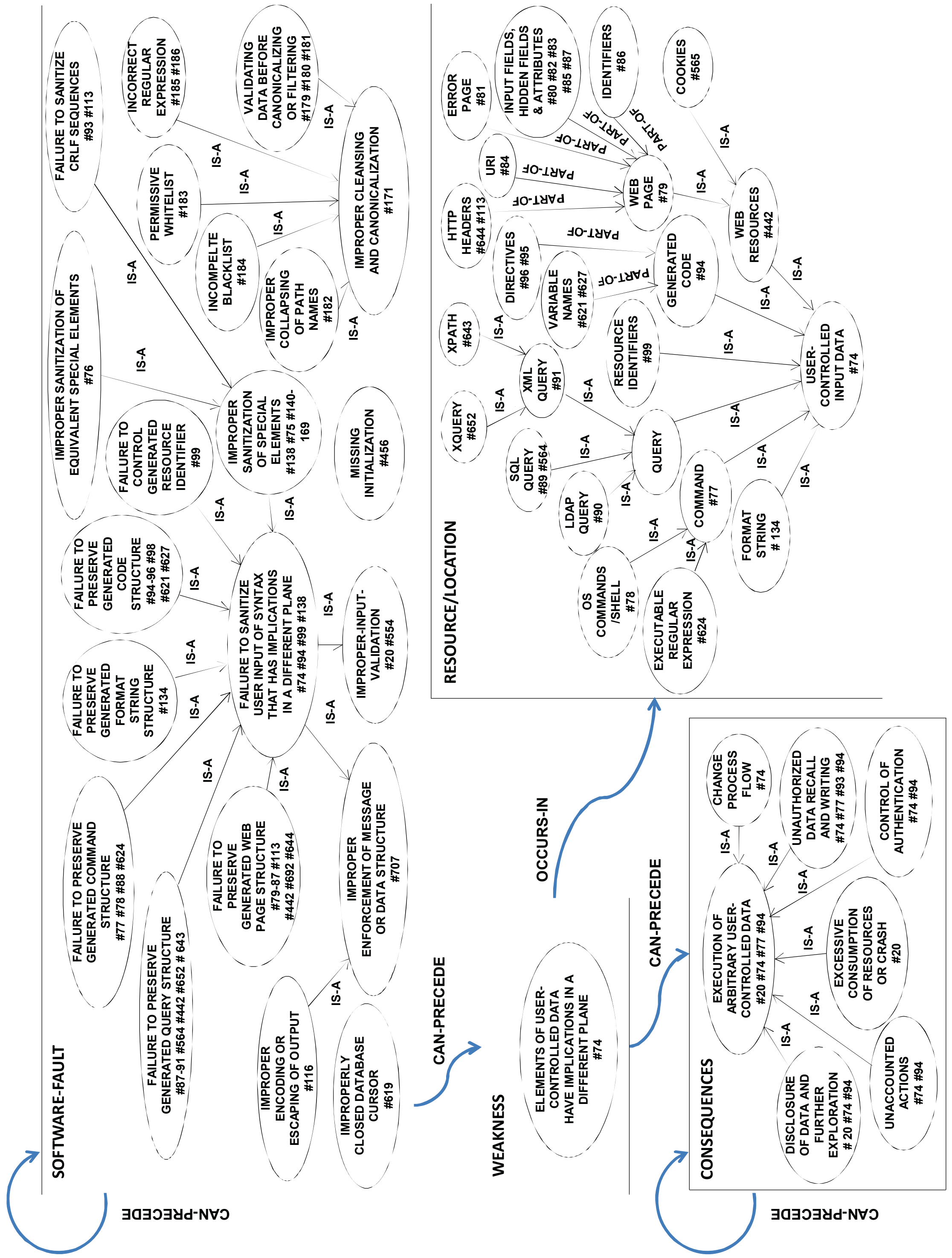
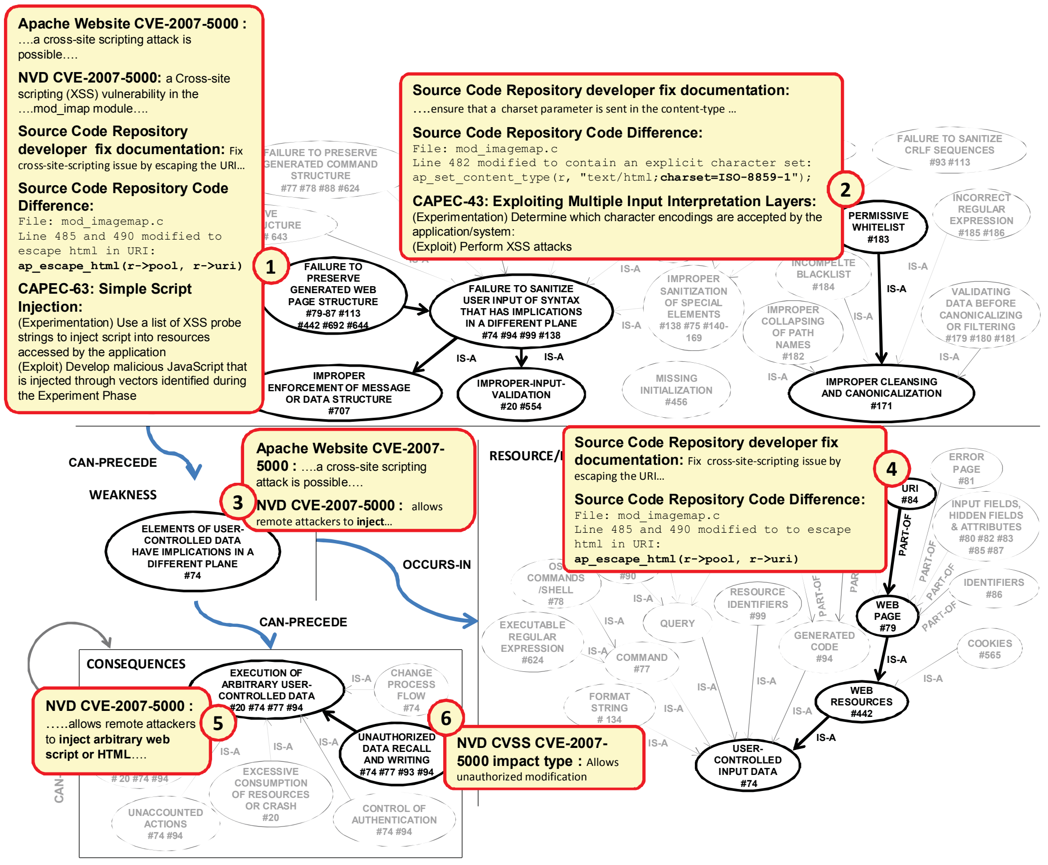

SOFTWARE-FAULT
FAILURE TO SANITIZE CRLF SEQUENCES #93 #113
INCORRECT REGULAR EXPRESSION #185 #186
VALIDATING DATA BEFORE CANONICALIZING OR FILTERING #179 #180 #181
IMPROPER SANITIZATION OF EQUIVALENT SPECIAL ELEMENTS #76
PERMISSIVE WHITELIST #183
INCOMPELTE BLACKLIST #184
IMPROPER COLLAPSING OF PATH NAMES #182
IMPROPER CLEANSING AND CANONICALIZATION #171
FAILURE TO CONTROL GENERATED RESOURCE IDENTIFIER #99
IMPROPER SANITIZATION OF SPECIAL ELEMENTS #138 #75 #140-169
MISSING INITIALIZATION #456
FAILURE TO PRESERVE GENERATED CODE STRUCTURE #94-96 #98 #621 #627
FAILURE TO PRESERVE GENERATED FORMAT STRING STRUCTURE #134
FAILURE TO SANITIZE USER INPUT OF SYNTAX THAT HAS IMPLICATIONS IN A DIFFERENT PLANE #74 #94 #99 #138
IMPROPER-INPUT-VALIDATION #20 #554
FAILURE TO PRESERVE GENERATED COMMAND STRUCTURE #77 #78 #88 #624
FAILURE TO PRESERVE GENERATED QUERY STRUCTURE #87-91 #564 #442 #652 # 643
FAILURE TO PRESERVE GENERATED WEB PAGE STRUCTURE #79-87 #113 #442 #692 #644
IMPROPER ENFORCEMENT OF MESSAGE OR DATA STRUCTURE #707
IMPROPER ENCODING OR ESCAPING OF OUTPUT #116
IMPROPERLY CLOSED DATABASE CURSOR #619
IS-A
CAN-PRECEDE
RESOURCE/LOCATION
ERROR PAGE #81
INPUT FIELDS, HIDDEN FIELDS & ATTRIBUTES #80 #82 #83 #85 #87
IDENTIFIERS #86
COOKIES #565
URI #84
HTTP HEADERS #644 #113
WEB PAGE #79
WEB RESOURCES #442
DIRECTIVES #96 #95
VARIABLE NAMES #621 #627
GENERATED CODE #94
XPATH #643
RESOURCE IDENTIFIERS #99
XML QUERY #91
XQUERY #652
USER-CONTROLLED INPUT DATA #74
SQL QUERY #89 #564
QUERY
LDAP QUERY #90
COMMAND #77
FORMAT STRING # 134
OS COMMANDS /SHELL #78
EXECUTABLE REGULAR EXPRESSION #624
PART-OF
OCCURS-IN
WEAKNESS
ELEMENTS OF USER-CONTROLLED DATA HAVE IMPLICATIONS IN A DIFFERENT PLANE #74
CONSEQUENCES
CHANGE PROCESS FLOW #74
UNAUTHORIZED DATA RECALL AND WRITING #74 #77 #93 #94
CONTROL OF AUTHENTICATION #74 #94
EXECUTION OF ARBITRARY USER-CONTROLLED DATA #20 #74 #77 #94
EXCESSIVE CONSUMPTION OF RESOURCES OR CRASH #20
DISCLOSURE OF DATA AND FURTHER EXPLORATION # 20 #74 #94
UNACCOUNTED ACTIONS #74 #94

Figure 8 - Injection Semantic Template

The injection semantic template was put to work to help study the artifacts available for the reported cross-site scripting vulnerability CVE-2007-5000 [40] in the Apache Web Server project, which are scattered across multiple repository sources. These sources include the NVD CVE database; Apache Website; Apache change history in the SVN software repository; source code versions; and related CAPECs. The semantic template allows researchers or practitioners to parameterize the natural language vulnerability description, change history, source code changes and other artifacts so as to fit the developer mental model of a certain injection weakness. In addition, the semantic template allows extrapolating the missing information, if there is any, from the vulnerability artifacts. Figure 8 showed clearly how to neatly classify the information from different sources which including NVD, apache website and source code changes. From description pieces of relevant vulnerabilities and source code, we located the software fault-related information pieces "negative content-length header", which leads to remote arbitrary code execution or large amount of data to be copied, as weaknesses existing in the software system, and this vulnerability occurs in the "heap-based" buffer, then its consequence could be "denial of service" or "remote arbitrary code execution". To annotate each occurrence of vulnerability, experts' manual searching and comprehension are necessary, not only for the natural language descriptions from all repository sources, but also the source code differences before and after the fix. In the following section, a semi-automatic annotation methodology will be introduced to alleviate the labor for experts on this manual work.

The semantic template provides intuitive visualization capabilities for the collected vulnerability artifacts. In Figure 9, the vulnerability artifacts related to

CVE-2007-5000 “trigger” and help “navigate” the concepts in the injection semantic template by relevant information pieces. Such an annotated semantic template allows developers to reason about the vulnerability, possible attack vectors (CAPECs), and the adequacy of performed fixes. It allows stakeholders in the SDLC to consume technical details with relative ease and guided explanation.



1

Apache Website CVE-2007-5000 : ….a cross-site scripting attack is possible….

NVD CVE-2007-5000: a Cross-site scripting (XSS) vulnerability in the ….mod_imap module….

Source Code Repository developer fix documentation: Fix cross-site-scripting issue by escaping the URI…

Source Code Repository Code Difference:

```
File: mod_imagemap.c
Line 485 and 490 modified to
escape html in URI:
ap_escape_html(r->pool, r->uri)
```

CAPEC-63: Simple Script Injection:
(Experimentation) Use a list of XSS probe strings to inject script into resources accessed by the application
(Exploit) Develop malicious JavaScript that is injected through vectors identified during the Experiment Phase

2

Source Code Repository developer fix documentation:
….ensure that a charset parameter is sent in the content-type …

Source Code Repository Code Difference:

```
File: mod_imagemap.c
Line 482 modified to contain an explicit character set:
ap_set_content_type(r, "text/html;charset=ISO-8859-1");
```

CAPEC-43: Exploiting Multiple Input Interpretation Layers:
(Experimentation) Determine which character encodings are accepted by the application/system:
(Exploit) Perform XSS attacks

3

Apache Website CVE-2007-5000 : ….a cross-site scripting attack is possible….

NVD CVE-2007-5000 : allows remote attackers to **inject**…

4

Source Code Repository developer fix documentation: Fix cross-site-scripting issue by escaping the URI…

Source Code Repository Code Difference:

```
File: mod_imagemap.c
Line 485 and 490 modified to to escape
html in URI:
ap_escape_html(r->pool, r->uri)
```

5

NVD CVE-2007-5000 : …..allows remote attackers to **inject arbitrary web script or HTML**….

6

NVD CVSS CVE-2007-5000 impact type : Allows unauthorized modification

FAILURE TO PRESERVE WEB PAGE STRUCTURE #79-87 #113 #442 #692 #644
FAILURE TO SANITIZE USER INPUT OF SYNTAX THAT HAS IMPLICATIONS IN A DIFFERENT PLANE #74 #94 #99 #138
IMPROPER ENFORCEMENT OF MESSAGE OR DATA STRUCTURE #707
IMPROPER-INPUT-VALIDATION #20 #554
FAILURE TO PRESERVE GENERATED COMMAND STRUCTURE #77 #78 #88 #624
FAILURE TO SANITIZE CRLF SEQUENCES #93 #113
PERMISSIVE WHITELIST #183
INCORRECT REGULAR EXPRESSION #185 #186
INCOMPLETE BLACKLIST #184
IMPROPER SANITIZATION OF SPECIAL ELEMENTS #138 #75 #140-169
IMPROPER COLLAPSING OF PATH NAMES #182
VALIDATING DATA BEFORE CANONICALIZING OR FILTERING #179 #180 #181
MISSING INITIALIZATION #456
IMPROPER CLEANSING AND CANONICALIZATION #171
IS-A
CAN-PRECEDE
WEAKNESS
ELEMENTS OF USER-CONTROLLED DATA HAVE IMPLICATIONS IN A DIFFERENT PLANE #74
OCCURS-IN
RESOURCE/
CONSEQUENCES
EXECUTION OF ARBITRARY USER-CONTROLLED DATA #20 #74 #77 #94
CHANGE PROCESS FLOW #74
UNAUTHORIZED DATA RECALL AND WRITING #74 #77 #93 #94
EXCESSIVE CONSUMPTION OF RESOURCES OR CRASH #20
UNACCOUNTED ACTIONS #74 #94
CONTROL OF AUTHENTICATION #74 #94
OS COMMANDS /SHELL #78
EXECUTABLE REGULAR EXPRESSION #624
QUERY
COMMAND #77
FORMAT STRING # 134
RESOURCE IDENTIFIERS #99
GENERATED CODE #94
ERROR PAGE #81
URI #84
INPUT FIELDS, HIDDEN FIELDS & ATTRIBUTES #80 #82 #83 #85 #87
IDENTIFIERS #86
COOKIES #565
WEB PAGE #79
WEB RESOURCES #442
USER-CONTROLLED INPUT DATA #74
PART-OF

Figure 9 - Annotated CVE-2007-5000 by Injection Semantic Template

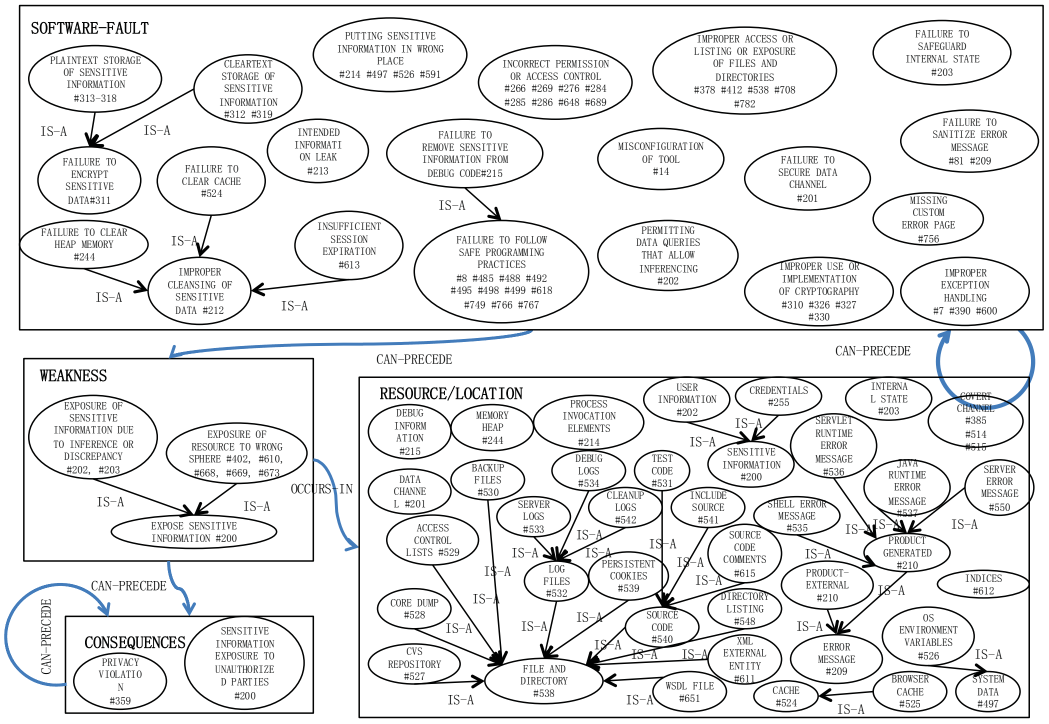

SOFTWARE-FAULT
PLAINTEXT STORAGE OF SENSITIVE INFORMATION #313-318
CLEARTEXT STORAGE OF SENSITIVE INFORMATION #312 #319
PUTTING SENSITIVE INFORMATION IN WRONG PLACE #214 #497 #526 #591
INCORRECT PERMISSION OR ACCESS CONTROL #266 #269 #276 #284 #285 #286 #648 #689
IMPROPER ACCESS OR LISTING OR EXPOSURE OF FILES AND DIRECTORIES #378 #412 #538 #708 #782
FAILURE TO SAFEGUARD INTERNAL STATE #203
FAILURE TO SANITIZE ERROR MESSAGE #81 #209
FAILURE TO ENCRYPT SENSITIVE DATA#311
FAILURE TO CLEAR CACHE #524
INTENDED INFORMATION LEAK #213
FAILURE TO REMOVE SENSITIVE INFORMATION FROM DEBUG CODE#215
MISCONFIGURATION OF TOOL #14
FAILURE TO SECURE DATA CHANNEL #201
MISSING CUSTOM ERROR PAGE #756
FAILURE TO CLEAR HEAP MEMORY #244
IMPROPER CLEANSING OF SENSITIVE DATA #212
INSUFFICIENT SESSION EXPIRATION #613
FAILURE TO FOLLOW SAFE PROGRAMMING PRACTICES #8 #485 #488 #492 #495 #498 #499 #618 #749 #766 #767
PERMITTING DATA QUERIES THAT ALLOW INFERENCING #202
IMPROPER USE OR IMPLEMENTATION OF CRYPTOGRAPHY #310 #326 #327 #330
IMPROPER EXCEPTION HANDLING #7 #390 #600
IS-A
CAN-PRECEDE
WEAKNESS
EXPOSURE OF SENSITIVE INFORMATION DUE TO INFERENCE OR DISCREPANCY #202, #203
EXPOSURE OF RESOURCE TO WRONG SPHERE #402, #610, #668, #669, #673
EXPOSE SENSITIVE INFORMATION #200
OCCURS-IN
CONSEQUENCES
PRIVACY VIOLATION #359
SENSITIVE INFORMATION EXPOSURE TO UNAUTHORIZED PARTIES #200
RESOURCE/LOCATION
DEBUG INFORMATION #215
MEMORY HEAP #244
PROCESS INVOCATION ELEMENTS #214
USER INFORMATION #202
CREDENTIALS #255
INTERNAL STATE #203
COVERT CHANNEL #385 #514 #515
SERVLET RUNTIME ERROR MESSAGE #536
SENSITIVE INFORMATION #200
DATA CHANNEL #201
BACKUP FILES #530
DEBUG LOGS #534
TEST CODE #531
JAVA RUNTIME ERROR MESSAGE #537
SERVER ERROR MESSAGE #550
ACCESS CONTROL LISTS #529
SERVER LOGS #533
CLEANUP LOGS #542
INCLUDE SOURCE #541
SHELL ERROR MESSAGE #535
PRODUCT GENERATED #210
LOG FILES #532
PERSISTENT COOKIES #539
SOURCE CODE COMMENTS #615
PRODUCT-EXTERNAL #210
INDICES #612
CORE DUMP #528
DIRECTORY LISTING #548
SOURCE CODE #540
OS ENVIRONMENT VARIABLES #526
CVS REPOSITORY #527
FILE AND DIRECTORY #538
XML EXTERNAL ENTITY #611
ERROR MESSAGE #209
WSDL FILE #651
CACHE #524
BROWSER CACHE #525
SYSTEM DATA #497

Figure 10 – Information Exposure Semantic Template

A draft version of the Information Exposure semantic template is also completed as shown in Figure 10. We leave the construction of the semantic template for the remaining vulnerability category identified in the trend analysis, Denial of Service, as future work.

Comparing the three semantic templates constructed thus far reveals some distinct characteristics among the respective categories. Concepts in the buffer overflow vulnerabilities are concentrated on software faults, indicating many coding errors which lead to accesses and writes outside of permitted buffer boundaries. Injection vulnerabilities are concentrated on software faults and resources, indicating interactions between faults and specific types of resources resulting in different variations of this vulnerability. Exposure vulnerabilities are concentrated on resources, indicating the variety of resources whose data can be inadvertently disclosed. We suspect that Denial of Service could help to observe vulnerabilities from another perspective: Consequences. This category of vulnerabilities can be preceded by different types of software faults and presented as different weaknesses in software system, but they share the same consequence as their tag: Denial of Service or crash of certain process.

5.5 Ontology Representation of the Semantic Template

From the Buffer Overflow semantic template, we constructed an ontology using the semantic template concepts as ontology concepts. Figure 11 shows vulnerability CAN-2004-0492 as a set of instances stored in the Buffer Overflow ontology. As showed in the picture, it is clear that this buffer overflow vulnerability is caused by an invalid or negative length content header, which is annotated as an improper input validation described in CWE-20. The weakness for this vulnerability is that large amount of data was copied, which is annotated by three layers of concepts, bottom up as access and out-of-bounds write, failure to constrain operations within the bounds of a memory buffer and improper access of indexable resource. The resource in which this vulnerability occurred is a heap in proxy_util.c for mod_proxy, which is annotated as a heap-based buffer and finally annotated buffer in the high level. The consequences mentioned in this vulnerability include denial of service and remote code execution, which is annotated as a write-what-where condition in this ontology. By this ontology-based view, the natural language description and source code of vulnerability instances is converted into a semantic-based structure with hierarchy and more meaning, providing a clearer understanding for stakeholders.

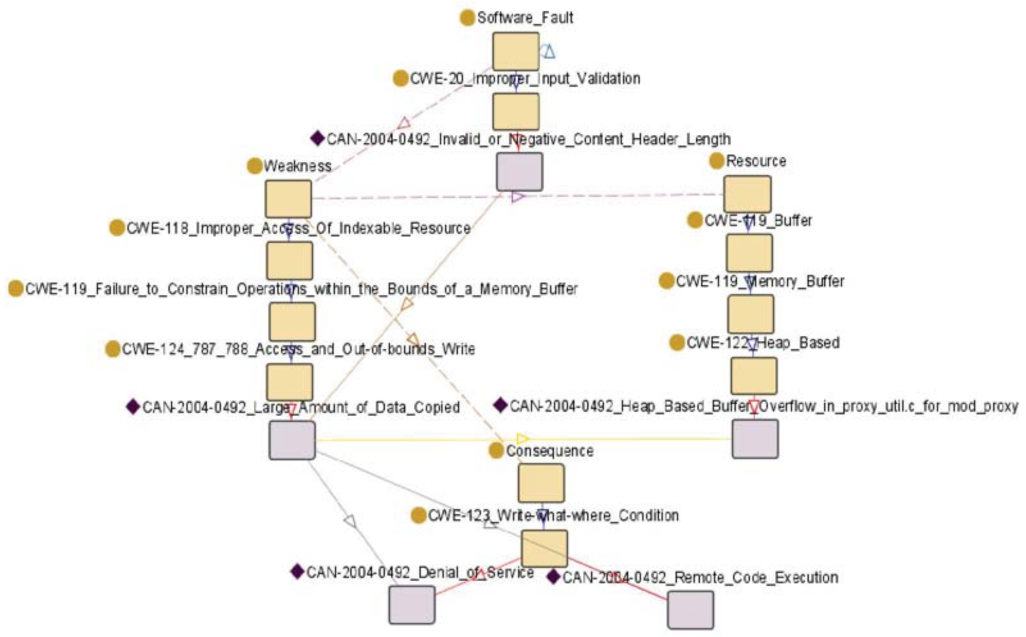

Software_Fault
CWE-20_Improper_Input_Validation
CAN-2004-0492_Invalid_or_Negative_Content_Header_Length
Weakness
Resource
CWE-118_Improper_Access_Of_Indexable_Resource
CWE-119_Buffer
CWE-119_Failure_to_Constrain_Operations_within_the_Bounds_of_a_Memory_Buffer
CWE-119_Memory_Buffer
CWE-124_787_788_Access_and_Out-of-bounds_Write
CWE-122_Heap_Based
CAN-2004-0492_Large_Amount_of_Data_Copied
CAN-2004-0492_Heap_Based_Buffer_Overflow_in_proxy_util.c_for_mod_proxy
Consequence
CWE-123_Write-what-where_Condition
CAN-2004-0492_Denial_of_Service
CAN-2004-0492_Remote_Code_Execution

Figure 11 - Modeling and Visualization of CAN-2004-0492 using Ontological Engineering Tool

6. Empirical Validation of the Effectiveness of Semantic Template to Study Vulnerabilities

Semantic templates provide a mental model for designers, developers and stakeholders from all the stages of software development lifecycle to study and understand security vulnerabilities. Once designed, several questions still remain unanswered regarding the ability of semantic templates to fulfill this promise. Questions such as: How effective are semantic templates to study vulnerabilities? Does its use reduce the time required to evaluate the weaknesses that led to the vulnerabilities? Does its use improve the accuracy of such an evaluation? We expect to answer these questions by conducting a controlled experiment in a lab environment.

In addition to lessons learned from a controlled experiment, we have also collected feedback comments from CWE editors, industry software assurance professionals, and software developers regarding the value of semantic templates in research, education and practice. Such feedback was collected at a forum where semantic templates were discussed in a working conference session.

6.1 Experiment [38]

To evaluate the effectiveness of semantic templates to sense the nature and possibility of an occurrence of vulnerability, 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 categories of mistakes that lead to same vulnerabilities. In this research work 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 with or without the relevant Semantic Templates, comparably, in time and accuracy.

6.1.1 Experimental Design

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

$H1_0$: 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.

And the corresponding alternative hypotheses are as follows:

$H1_a$: Subjects who use semantic templates spend less time for studying vulnerability reports compared to those who do not.

$H2_a$: Subjects who use semantic templates achieve better accuracy of understanding of vulnerabilities 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 the subjects' 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 [19]. In essence, there were 2 rounds of experiments. In Round 1, all subjects reviewed the vulnerability-related material with no additional aid. In Round 2, subjects in Group 2 were provided with semantic templates to assist in their study of vulnerabilities.

6.1.2 Pre-test

The subjects were randomly divided into 2 groups, and to reach a division as balanced as possible, the students were asked to review the CWE 1.6 specification with simple instructions only, and a pre-test sheet which includes demographic questions (as showed in Appendix A) was sent to the students one week before the experiment.

The pre-test questions are listed as below:

Please record the time you spent on this pre-test.

PQ1. Answer the following demographic questions on a scale of 0 – 10 (10 being the highest):

a. How would you rate your programming skill level?
b. How would you rate your experience in reading research/technical papers?

c. How would you rate your experience in writing research/technical papers?

PQ2. From the Research and Development Views of the CWE, how many entries are Buffer Overflow-related?

a. List their Ids.

b. List the different kinds of relationships between these entries.

PQ3. Determine if there is a relationship between the following pairs of CWE entries by tracing the relationships in the CWE 1.6 specification:

a. Is there any relationship between CWE-122 and CWE-20?

b. Is there any relationship between CWE-170 and CWE-682?

c. Is there any relationship between CWE-119 and CWE-101?

(If answer is yes, write down the relationship chain.)

PQ4. From the answer to question 2a, list the CWE entries that can precede the Buffer Overflow weaknesses.

PQ5. After answering questions 2-4, how would you rate your understanding of the Buffer Overflow weaknesses? (On a scale of 0–10.)

PQ6. Discuss your understanding of a "fault".

PQ7. Approximately, how long did this entire pre-test take?

Questions regarding programming skills, reading and writing technical papers, the students' ability to review and understand the vulnerability description and source code was evaluated. Simple questions concerning CWE entries were used to understand the student effort in reading and understanding the CWE specification.

After collecting and analyzing the answers to the pre-test questions, a profile of the students' knowledge and skill set and their degree of understanding the CWE specification was formed. Using these profiles, a randomized division of two groups was formed with the consideration of balancing knowledge, skills and the familiarity of CWE specifications between the two groups. As showed in Table 1, the results to PQ1a, PQ1b, PQ1c and PQ5 were tested by Shapiro–Wilk test [29] to verify the normal distribution of data.

Table 1 - W and p-values of Shapiro–Wilk test for Pre-test data

	PQ1a	PQ1b	PQ1c	PQ5
W	0.9214	0.9211	0.9214	0.9344
p-value	0.0292	0.0286	0.0292	0.0645

Based on the W value and p-value, except for the answers to PQ5, other pre-test data is not normally distributed, so instead of t-test, we test this pre-test answer data set by the Wilcoxon test in which dataset are not required to be normally distributed.

6.1.3 Settings and Materials

The experimental objects to be studied consisted of selected buffer overflow vulnerabilities reported in the Apache Web Server project. We selected the buffer overflow vulnerability for this experiment because it was relatively well known (paradoxically, it also occurs the most often). This was reinforced by the results of pre-test for the subjects who indicated that most of them have certain basic understanding of buffer overflow. From the 14 vulnerability reports that were unanimously voted as buffer overflow weaknesses by the investigators (Dissertation Committee Co-Chairs and myself), 6 (3 vulnerability reports to study for each round) were selected for this empirical study. The 6 reports were carefully chosen to account for the limited time that subjects had to examine the materials in each round of the experiment. To keep the time, difficulty and a balanced work load between the two rounds of experiment, statistical data such as the number of source code files relevant to each vulnerability report, word count of natural language description, and number of lines of source code in the revisions to the vulnerabilities were collected. Using this data two sets of vulnerability for the two experiment rounds were formed. The resulting 2 sets of three vulnerability reports were such that they can be easily reviewed and analyzed in the allotted 60 minutes for each round.

The materials given to the subjects for each vulnerability report used in the experiment include:

1. Vulnerability ID (a standard CVE identifier)
2. Descriptions from both the Apache website and the National Vulnerability Database (NVD)
3. Change description logs from the Apache Subversion repository, and
4. The source code differences before and after the vulnerability was fixed.
5. A hyperlinked CWE .pdf document.

For each vulnerability report identified by a CVE number, the subjects marked their start and end time points, and answered the following questions:

Question 1. List the related CWE entries for the CVE investigated.

Question 2. What faults may have led to the CVE?

Question 3. 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?)

Question 4. How can we train developers to avoid this problem?

These questions are primarily geared towards the investigation of a vulnerability and thus, as neutral as possible to limit the extent to which students can sense the research questions from them.

6.1.4 Variables

The experiment manipulated these three independent variables:

1. Group - refers to the group assigned (1 or 2, group 2 was given Buffer Overflow Semantic Template in the second round).
2. Round - refers to the experiment round (1 or 2).
3. Vulnerability ID - the ID of those vulnerabilities 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 (1-10 scale) from pre-test:

1. Programming skill level (based on the answer to PQ1a)
2. Reading comprehension and writing skill levels - ability to read and write technical English documents. (based on the answers to PQ1b and PQ1c)

These dependent variables were collected for each vulnerability report from each subject:

1. **Time to complete** - time (in minutes) to study and complete answering the questions about the corresponding vulnerability.

2. **CWE identification accuracy** – CWE entries correctly identified as related to the vulnerability (measured by precision and recall).
3. **Fault description accuracy** - a score (scale of 1-5) on the accuracy of the identification of the software fault that led to the vulnerability.
4. **Failure description accuracy** - a score (scale of 1-5) on the accuracy of the description of the nature of the vulnerability (the manifested problem, the resources impacted and the consequences).

Time to complete is used to test H1, while CWE identification accuracy is used to test H2, then the other two, fault description accuracy and failure description accuracy are also collected to give us an additional insight of subjects' understanding of vulnerabilities. The set of relevant CWEs entries for each vulnerability report (answers to Question 1) was determined by consensus among the three experts (the Dissertation Committee Co-Chairs and myself) prior to the experiment. To achieve reliability of the evaluation scores for the last two measures (answers to Question 2 and Question 3), each expert 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.

6.1.5 Preparation and Conduct of Experiment

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

Rounds 1 and 2 were conducted on 11/29/2010 and 12/01/2010, respectively. The experiment was conducted in a supervised laboratory environment with no Internet access. Each subject was initially given the question sheet, related documents and information for the first vulnerability report. Upon finish answering all the questions for that vulnerability, the subject submitted the completed question sheet, with the time marked, and then the subject proceeded to the next vulnerability. In this way, we ensured that all times reported are accurate to the minute.

For Round 2, Group 2 was assigned to use semantic templates. A brief 15-minute tutorial on the use of semantic templates was provided to Group 2 immediately before Round 2 in a separate lab. 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 assigned to a project.

6.1.6 Results

Based on the fact that part of the experiment results are subjective, the raw data was tested by the Shapiro–Wilk test [29] before any further analysis, which is designed to test the null hypothesis that a sample $x_1, ..., x_n$ came from a normally distributed population. The results for the "Completion Time" are shown in Table 2.

Table 2 - W and p-values of Shapiro–Wilk test for Time

Round 1	W	(1-1) 0.8099	(1-2) 0.9338	(1-3) 0.8624
	p-value	(2-1) 0.0001	(1-2)0.0618	(1-3)0.0011
Round 2	W	(2-1) 0.8645	(2-2) 0.8872	(2-3) 0.8945
	p-value	(2-1) 0.0013	(2-2)0.0041	(2-3)0.0062

The results showed that except for the completion time for vulnerability 1-2 in round 1, other time data sets are not normally distributed. Based on this result, the t-test will not be effective for our data, so the Wilcoxon signed-rank test was chosen to test our experiment null hypothesis $H1_0$.

Before presenting the individual results, a post-test (multiple linear regression) was executed to evaluate the level of contribution from the pre-test variables to the experiment results, so that the randomized splitting of two groups can be verified partially. There were no significant contributions discovered from the pre-test results to the experiment results.

6.1.6.1 Time to Completion

Although the student subjects worked on the same tasks, it still took widely different time intervals for them to analyze the vulnerability reports and answer experiment questions. By observation, different students hold different characteristics and preferred strategies: some liked to spend more time searching CWE documents; some preferred spending more time reading vulnerability descriptions; some preferred starting by scrutinizing the initial tasks, then finish the later instances much faster and some almost evenly distribute their time on the three vulnerability instances. The completion time for one vulnerability report by different students ranged from 3 minutes to 55 minutes.

The box-plots in Figure 12 depict the time it took for each subject from both groups to analyze each vulnerability report during the two rounds, presenting the lowest, highest and the median. The top row shows the comparisons between the two groups for Round 1 when neither group used semantic templates. The similar medians in all three reports suggest that originally the two groups were comparable in their capability to understand and analyze the vulnerability instances when measured by completion time.

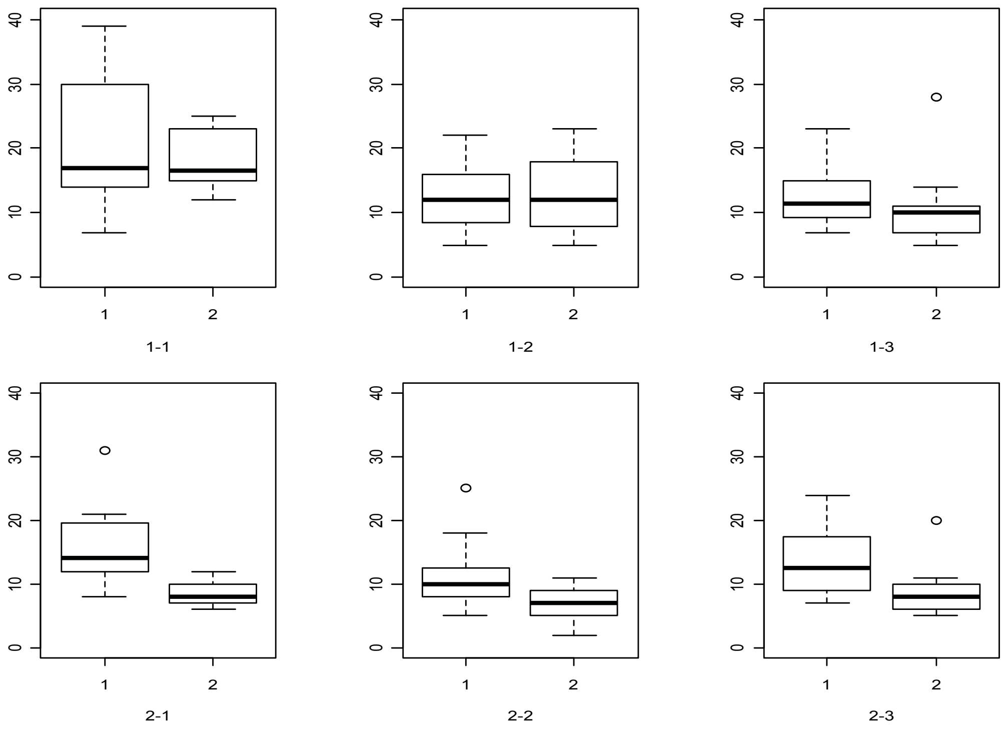


Figure 12 - Time to completion (minutes) per vulnerability

The bottom row in Figure 12 shows the comparisons with Group 2 using semantic templates. The noticeably lower medians for group 2 in the bottom row suggests that Buffer Overflow semantic template has a noticeable effect on shortening the time to complete understanding and analyzing every vulnerability report. Based on the fact that time data was not normally distributed, the t-test cannot be performed to determine the significance of this difference in time data (as determined by the Shapiro-Wilk [29] test for normality shown in Table 3). The differences of time spent between two groups in both rounds were statistically verified by the Wilcoxon signed-rank test. The W and p-values are shown in Table 3 (The values with $p < 0.05$ are in boldface). It is obvious that in all the vulnerability reports in round 2 and round 1 question 3, the critical values are less than .05 and rejected the $H0: \theta = 0$, which means that in 1-3, 2-1, 2-2 and 2-3, the students of group 1 spent significantly more time on their answers compared with group 2. (1-3 p-value is very near to the .05 threshold, as a not very critical value)

The results manifest that for Round 1, there was no difference in time 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.

Table 3 - W and p-values of Wilcoxon signed-rank test (alternative hypothesis is group1 value is greater than group2 value) for average Time data

Round 1	W	(1-1) 114.5	(1-2) 107	(1-3) 152.5
	p-value	(1-1) 0.4668	(1-2)0.5907	**(1-3)0.0473**
Round 2	W	(2-1) 203	(2-2) 176.5	(2-3) 190
	p-value	**(2-1)7.847e-05**	**(2-2)0.0037**	**(2-3)0.0006**

To confirm the result, we also calculated the average time spent for each subject of both groups in the two rounds and drew the boxplot for the average time difference for all the subjects as t2-t1 (average time spent in three reports of round 2 minus average time spent in three reports of round 1), measured as the improvement for each student from round 1 to round 2. The resulted boxplot in Figure 13 shows that group 2 had a greater time reduction (improvement) compared with group 1. To further prove it, a Wilcoxon signed-rank test was conducted suggesting that the average time difference (as a negative value) for group 1 is greater than the average time difference for group 2, it also means that group 2 has a greater time reduction from round 1 to round 2. Thus the null hypothesis $H1_0$ is rejected and H1a is accepted.

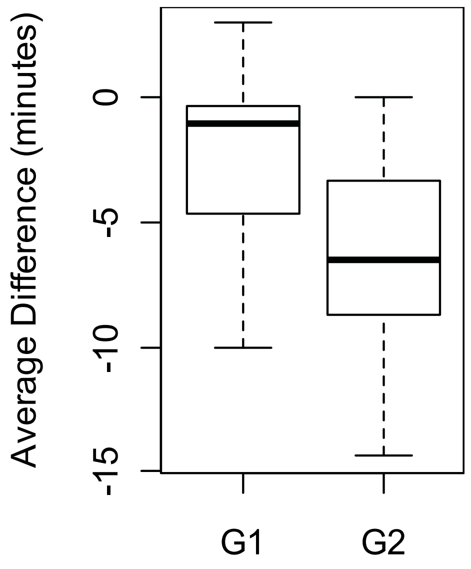


Figure 13 - Difference of Average Time for subjects of two groups from round 1 to round 2 (W = 179.5, p-value = 0.002667)

6.1.6.2 CWE Identification Accuracy

The CWE identification accuracy estimates how well the subjects identified the correct set of CWEs that are relevant to a certain CVE entry. This variable depends on the answer to Question 1 for each vulnerability report. The accuracy was measured using precision and recall metrics (as defined in Appendix A), against expert identified standard answer sets. The expert answers, as the experiment oracle, are listed in Table 4. The rows correspond to the concept groups in the semantic templates and the columns correspond to the vulnerability IDs used in the experiment.

Table 4 – Expert Oracle

	2004-0492	2004-0493	2005-1268	2009-0023	2009-2412	2010-0010
Fault	20, 130	129-789, 131, 194-195-196, 682	193, 682	20, 128, 191, 192, 194-195-196, 682	20, 128, 190-680, 682	128, 190-680, 192, 682
Weakness	118, 119, 124, 787, 788	118, 119, 124, 787, 788	118, 119, 124, 787, 788	118, 119, 124, 787, 788	118, 119, 124, 787, 788	118, 119, 124, 787, 788
Resource	118, 119, 122	118, 119, 122, 129	118, 119, 121	118, 119, 122	118, 119, 122	118, 119, 122
Consequence	123	123, 789	123	123	123	123

The Shapiro-Wilk tests indicated that both precision and recall values were not normally distributed, hence t-tests cannot be used, so we used the Wilcoxon signed-rank test to test whether there is any improvement for Group 2. The box-plots in Figure 14 depict the precision of Question 1 answer for each subject from both groups for each vulnerability report during the two rounds, presenting the lowest, highest and the median. The top row shows the comparisons between the two groups for Round 1 when neither group used semantic templates, which indicated that originally students from group 2 underperformed compared with group1 in the first two vulnerability reports and share a similar precision in the third vulnerability report. But in the bottom row, it appears that the usage of semantic templates did not have any significant impact on precision. However, in Round 2 using semantic templates, group 2 performed at par with group 1, making up for the difference in Round 1.

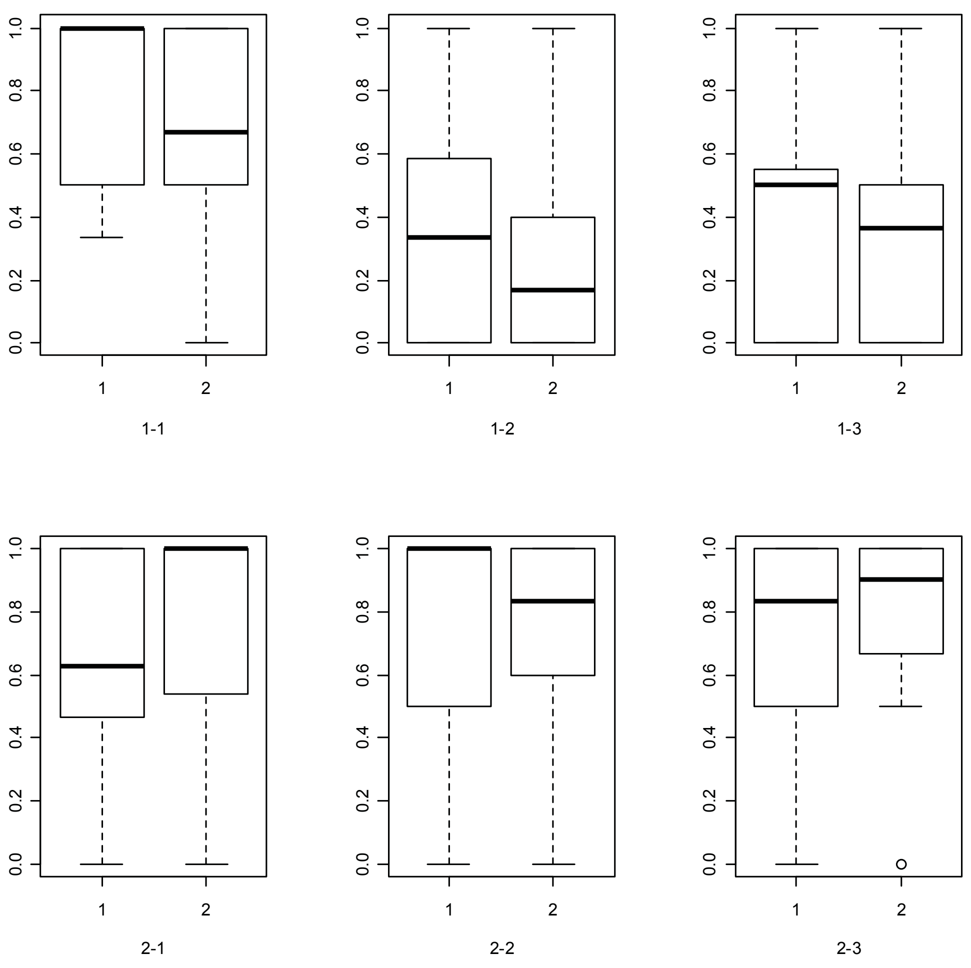


Figure 14 - Question 1 answer precision per vulnerability

The difference of precision on Question 1 answer between two groups was statistically verified by the Wilcoxon signed-rank test with the alternative hypothesis that group 1 precision is less than group 2 precision. The W and p-values are shown in Table 5. It is obvious that none of the p value is significant, which means and Question 1 answer precision of group 1 is not less than that of group 2 considering the individual reports.

Table 5 - W and p-values of Wilcoxon signed-rank test for Question 1 answer precision data

Round 1	W	(1-1) 140.5	(1-2) 137.5	(1-3) 127.5
	p-value	(1-1) 0.897	(1-2) 0.8693	(1-3) 0.7555
Round 2	W	(2-1) 90	(2-2) 120.5	(2-3) 99.5
	p-value	(2-1) 0.1685	(2-2) 0.6538	(2-3) 0.2966

We also calculated the average precision for each subject of both groups in the two rounds and drew the boxplot for the average precision difference for all the students as p2-p1, measured as the improvement for each student from round 1 to round 2. The resulted plot in Figure 15 showed that group2 had a greater improvement on their precision of Question 1 answers with a higher median and higher boundary, compared with group 1. To further prove it, a Wilcoxon signed-rank test was conducted and with a p-value < 0.05, it showed that the null hypothesis was rejected and the average precision difference for group 1 is less than the average precision difference for group 2, it also means that group 2 has a greater precision improvement from round 1 to round 2.

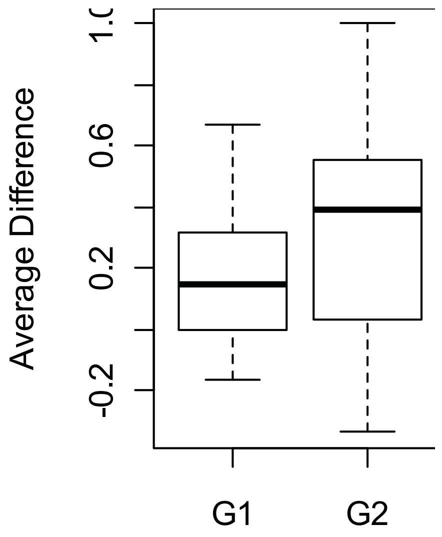


Figure 15 - Average Question 1 precision for two groups from round 1 to round 2 (), W = 76, p-value = 0.06978)

The box-plots in Figure 16 depicts the recall of Question 1 answer for each subject from both groups for each vulnerability report during the two rounds, presenting the lowest, highest and the median. The top row shows the comparisons between the two groups for Round 1 when neither group used semantic templates, which indicated that originally there were no significant differences between the two groups. But in the bottom row, it seemed that group 2 performed much better than group 1, indicating that the usage of semantic templates have significant impact on recall.

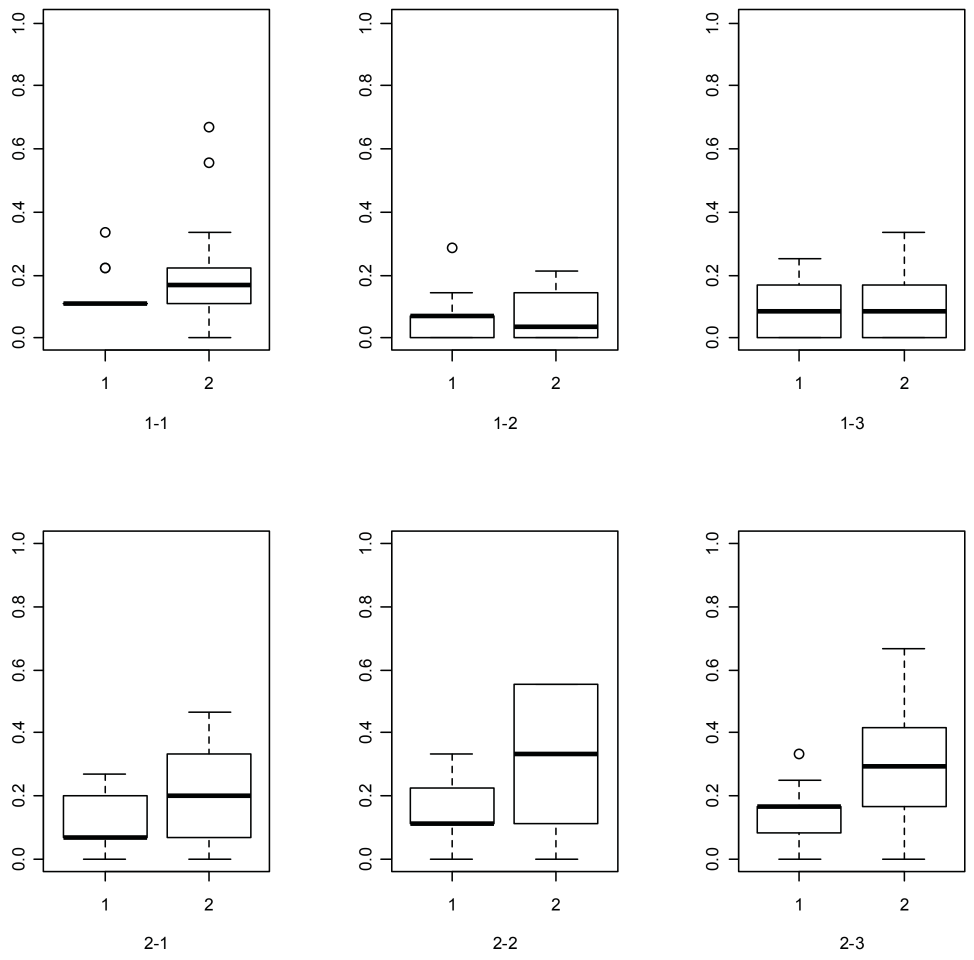


Figure 16 - Question 1 answer recall per vulnerability

The difference of recall on Question 1 answer between two groups was statistically verified by the Wilcoxon signed-rank test with the alternative hypothesis that group 1 recall is less than group 2 recall. The W and p-values are shown in Table 6 (significant p-value are boldface). It is obvious that in round 1, none of the p value is significant, while in round 2 all the p values are significant, which means that in round 2, the null hypothesis can be rejected, and the recall of question 1 answer is significantly less than that of group2.

Table 6 - W and p-values of Wilcoxon signed-rank test for Question 1 answer recall data

Round 1	W	(1-1) 81.5	(1-2) 119.5	(1-3) 104.5
	p-value	(1-1) 0.0733	(1-2)0.6392	(1-3)0.3803
Round 2	W	(2-1) 67.5	(2-2) 62	(2-3) 47
	p-value	**(2-1)0.0308**	**(2-2)0.0162**	**(2-3)0.0030**

We also calculated the average recall for each subject of both groups in the two rounds and drew the boxplot for the average recall difference for all the students as r2-r1, measured as the improvement for each student from round 1 to round 2. The resulted plot in Figure 17 showed that group 2 had a greater improvement on their recall of Question 1 answers with the higher median and both boundaries, compared with group 1. To further prove it, a Wilcoxon signed-rank test was conducted and with a p-value < 0.01, it showed that the null hypothesis was rejected and the average recall difference for group 1 is less than the average recall difference for group 2, it also means that group 2 has a greater recall improvement from round 1 to round 2.

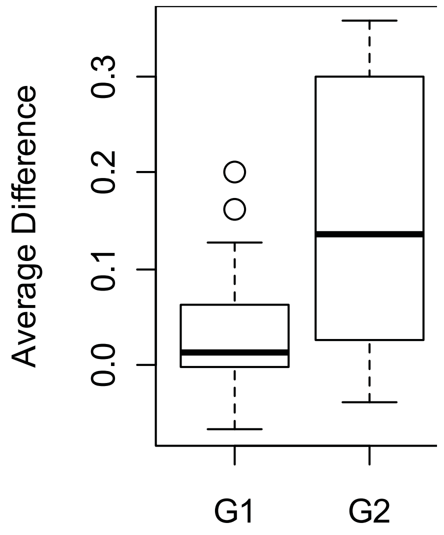


Figure 17 - Average Question 1 recall for two groups from round 1 to round 2 (W = 52, p-value = 0.006686)

These initial results indicate that semantic templates help in identifying the correct CWEs. Although the precision difference of CWE identification between two groups in round 2 is not significant, the improvement of group 2 is significantly greater than that of group 1. And in the recall of question 1 answer, group2 performed better compared with group1 in round 2, also the improvement of recall of group2 is greater than that of group1. Thus $H2_0$ can be rejected with respect to CWE identification accuracy and $H2_a$ is accepted.

Other than the individual precision and recall, we also collected the top five selected CWE entries from students and compared with expert annotation. The results are encouraging. Although we did not give students the instruction about including all the root nodes when selecting certain nodes, the precision relatively is rather high, and the recall is medium, as shown in Table 7.

Table 7 – Top Five Results of Students Experiment

	2004-0492	2004-0493	2005-1268	2009-0023	2009-2412	2010-0010
1	130	***122***	***193***	***124***	***122***	***122/190***/681
2	122	400	***121***	***122***	***119/190***	189
3	118	***119***	***119***	***787***	***20***	20
4	240	20/399/401	***682***	***196***	***118***	***119***/120/***680***
5	***20***/120	120/633/730	***118***/170	***119***	130	***192***/131/136 /197/704
Precision	66.7%	22.2%	83.3%	100%	83.3%	38.4%
Recall	44.4%	14.3%	55.6%	33.3%	41.7%	41.7%

6.1.7 Analysis of Open-ended Questions

In addition to test the hypotheses, we also ask the subjects questions about descriptions of the fault and failure. The answers to the open-ended questions Q2 and Q3 were rated to gauge how well the subjects understood the root cause and nature, respectively, of each occurrence of vulnerability. The answers were rated on an ordinal scale of 1-5 according to the following rubric:

1	Wrong or unexpected answer
2	Too vague to be considered right or wrong
3	Answer consistent with vulnerability description
4	Answer consistent with vulnerability description and provides insightful description
5	Detailed correct answer (reference to CWEs, Code sections identified, chain of events)

As Q2 and Q3 were subjective, all three researchers independently rated each answer. The inter-rater reliability was computed using Cronbach's alpha. The inter-rater reliability (see Table 8) was generally good for Q2 but less so for Q3 (alpha values above 0.7 indicate acceptable inter-rater reliability).

Table 8 - Inter-rater Reliability for Q2 and Q3 Answer

ID	Q2	Q3
1-1	0.8423978	0.616875
1-2	0.7636008	0.7623291
1-3	0.8241820	0.4148707
2-1	0.7558455	0.8198917
2-2	0.7347231	0.5561038
2-3	0.5586861	0.6509174

The modes of the three researchers' ratings were used as the values of the two dependent variables, Fault Identification Accuracy and Description Accuracy. The per-vulnerability comparison of the performance of the two groups indicate no significant differences in either round, as shown in Figures 18, 19 and Tables 9, 10. Based on these initial results, the effect of semantic templates was not large enough to be significant.

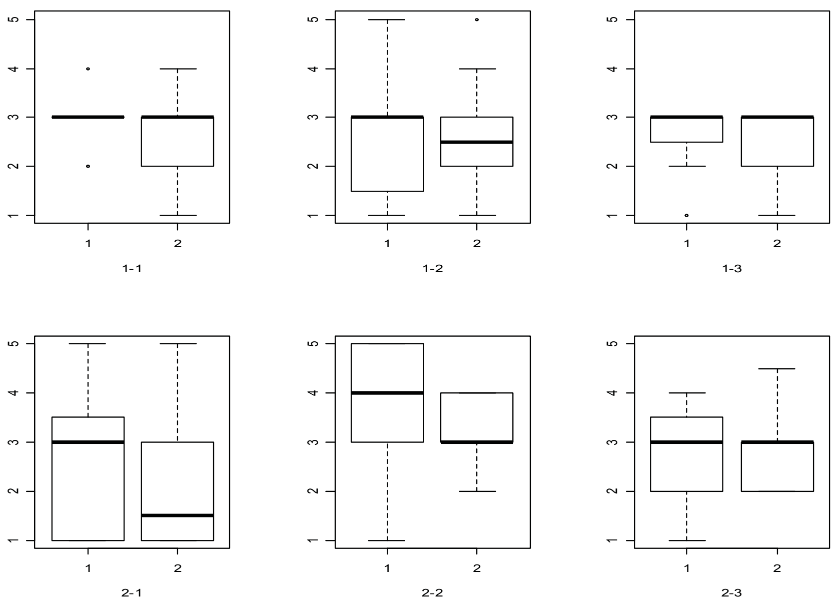


Figure 18 - Accuracy of Q2 per vulnerability

Table 9 - W and p-values of Wilcoxon signed-rank test (alternative hypothesis is group1 value is less than group2 value) for Question 2 answer

Round 1	W	(1-1) 122	(1-2) 102.5	(1-3) 120.5
	p-value	(1-1) 0.6721	(1-2) 0.353	(1-3) 0.6485
Round 2	W	(2-1) 127	(2-2) 163.5	(2-3) 133
	p-value	(2-1) 0.7427	(2-2) 0.9854	(2-3) 0.8164

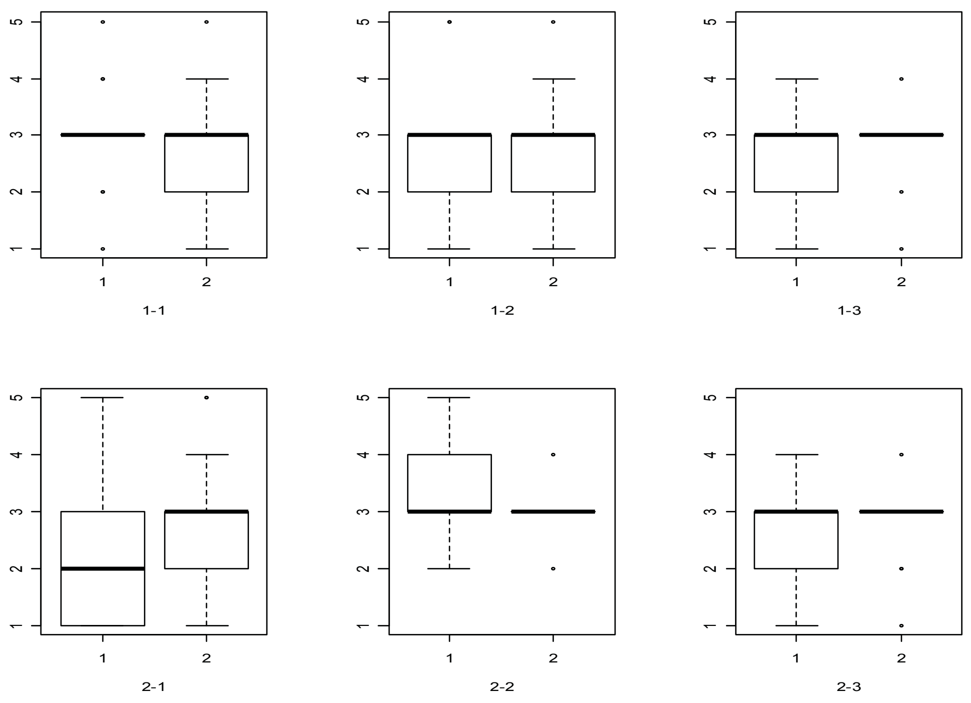


Figure 19 - Accuracy of Q3 per vulnerability

Table 10 - W and p-values of Wilcoxon signed-rank test (alternative hypothesis is group1 value is less than group2 value) for Question 3 answer

Round 1	W	(1-1) 130.5	(1-2) 108.5	(1-3) 109
	p-value	(1-1) 0.788	(1-2) 0.4497	(1-3) 0.4575
Round 2	W	(2-1) 97	(2-2) 134	(2-3) 129
	p-value	(2-1) 0.2721	(2-2) 0.8292	(2-3) 0.7684

6.1.7.1 Analysis of Variance

To understand these results, analysis of variance was performed to determine if other variables had a stronger effect. Because both dependent variables were ordinal scales, most statistical models, e.g., Gaussian, Poisson, etc., did not fit well. Instead, we transformed them into binary variables ("true" if variable >= 3 and "false" if variable < 3), and used logistic regression to compute the analysis of variance. We tested the effect of semantic template usage, round, and time to complete and found no significant contributions to either dependent variable. Interestingly, when we also account for precision and recall of CWE identification accuracy, recall is a significant contributor (see Tables 11 and 12). This indicates that participants who have relatively high CWE recall have higher probability of also scoring well on the accuracy questions. On the other hand, CWE precision had a significant though negative effect. Generally, participants achieve higher precision by identifying fewer CWEs and these CWEs turn out to be the valid ones. The negative result for precision indicates that recall is more important, that is, it is more important to identify the complete set of valid CWEs in order to satisfactorily answer the accuracy questions. Lastly, the presence of semantic templates did not positively contribute to higher probability of scoring well. In fact, for fault identification accuracy, it had a significant negative effect, though the contribution was not as strong as CWE recall.

Table 11 - Effects of experimental factors on Fault Identification Accuracy

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	1.371e-01	6.734e-01	0.204	0.838664
Round	2.982e-01	4.070e-01	0.733	0.463781
ST	-1.144	5.329e-01	-2.146	0.031841 *
Precision	-9.408e-01	5.091e-01	-1.848	0.064600
Recall	5.160	1.522	3.390	0.000698 ***
Time	5.841e-06	2.272e-02	2.57e-04	0.999795

Table 12 - Effects of experimental factors on Description Accuracy

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	1.65139	0.70258	2.350	0.0188 *
Round	-0.37776	0.39836	-0.948	0.3430
ST	0.43819	0.54717	0.801	0.4232
Precision	-1.10611	0.52308	-2.115	0.0345 *
Recall	3.56442	1.58974	2.242	0.0250 *
Time	-0.01717	0.02273	-0.755	0.4501

6.1.7.2 Impact on Terminology Usage

We also conjecture that if semantic templates can provide suitable mental models for understanding vulnerabilities, then we ought to see more usage of standard CWE language in describing faults and failures. With this in mind, we re-examined the open-ended answers in Round 2 to see if participants who used semantic templates ended up using the related terminologies. The answers were rated on an ordinal scale of 1-3 according to the following rubric:

1	No Standard terminology used
2	Some use of standard terminology
3	Consistent/Significant use of standard terminology

As before, the inter-rater reliability was computed using Cronbach's alpha. The inter-rater reliability for Q2 is acceptable while Q3 is borderline (see Table 13)

Table 13 - Inter-rater reliability for term usage

	Q2	Q3
2-1	0.7954545	0.6941106
2-2	0.8538117	0.674821
2-3	0.7226174	0.8056013

The boxplots in Figure 20 show that Group 2 tended to have a higher median than Group 1, but the difference (Wilcoxon-Mann-Whitney) is not statistically significant.

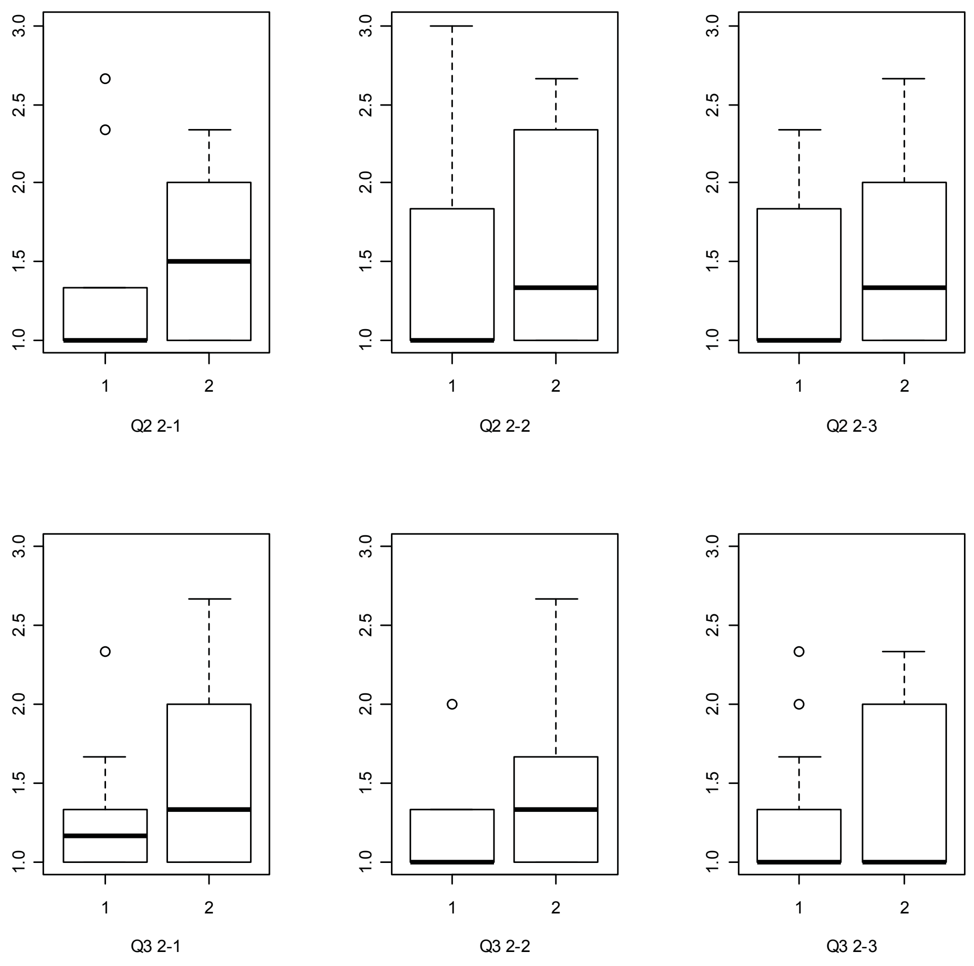


Figure 20 - Question 2 and 3 Answer Term Usage per Vulnerability

In summary, semantic templates did not improve the fault identification or description accuracy of the experiment participants. Though semantic templates did improve CWE recall and participants with higher CWE recall have higher probability of scoring better on the open-ended questions, this secondary effect was not enough for semantic templates to improve the accuracy scores. And while there is some improvement in usage of terminologies, the improvement is not statistically significant.

In hindsight, discovering faults and failures requires time and experience with the code. Semantic templates by themselves are not a root cause investigation tool, though experts can document known vulnerabilities and their root cause by annotating the templates as illustrated in the previous chapter.

6.1.8 Threats to Validity

The experiment was designed to mitigate as more threats to validity as possible. They will be introduced in the following sub-sections.

6.1.8.1 Threats to internal validity

Selection threats were mitigated by random 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. The two rounds of the experiment were conducted on two days with only one day between them diminished the history and maturation threats to a certain degree. Learning effects cannot be eliminated in this experiment, so two rounds and two sets of different vulnerabilities were used to avoid the learning effect by comparing the learning effects for each group. In the second round, the two groups of students were separated as much as possible (due to the limitation of laboratory seats, half of the treatment group was still in the same lab with the control group but with a line of computers as separation) to avoid the diffusion effect. To avoid the experimenter bias threats, each of the students was given a random ID by one of the experimenter who does not know any of the students.

6.1.8.2 Threats to external validity

As with other classroom experiments conducted with student subjects, generalization of results is an issue. First of all, students are not professionals and they are not familiar with the Apache product. We argue that this is not a big drawback as they would be close to the profile for a novice developer who requires more effective training. 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.

6.1.9 Discussion

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 precision and recall of relevant CWEs. Further insights into accuracy of understanding will be gained once we analyze the 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.

6.2 Expert Survey

Although semantic template were validated to be effective in improving the review time and CWE identification recall by the student experiment, it still needs the recognition and acceptance from the software engineering and software assurance communities for its value in research, education and practice.

The author attended International Conference on Software Engineering (ICSE) 2011 and presented the empirical study section as a research paper and poster. Feedback from the audience at this highly respected research venue was positive and valuable. During the 45-minute poster session, several professionals from industry, researchers and students showed interest in this research work. The questions most asked include the completeness and correctness of semantic template, the use of semantic templates for training, and the problem motivation. Researchers from the software assurance community liked the way semantic templates organize weakness knowledge, while the industry practitioners are interested in the possibility for semantic template to train novice developers and test engineers.

Committee chair Dr. Gandhi conducted a survey among the software assurance professionals in attendance at the 2011 Software Assurance Forum. He also presented the semantic template research at this forum. The survey feedback is listed in Appendix B. The overall feedback is encouraging, with four out of five respondents perceiving that semantic templates are very useful in studying weaknesses. They also stressed the need for more semantic templates in the future and a central repository to store and access the templates. The respondents also expressed what they liked and hated about semantic templates, which are the source of our improvements. They generally liked the clarity and separation of concepts, and the potential to training developers. The potential shortcoming pointed out includes the potentially high training overload, the possible instability of CWE as foundation, and the challenge in automatically annotating vulnerability data with concepts in the semantic template. We address the challenge of automatic annotation of vulnerability data in the next chapter. Finally, three of the five respondents showed interests in future collaboration on this research.

7. Semi-automatic Annotation of Vulnerabilities

Manual comprehension and annotation by experts is a series of time-consuming work which includes tedious collecting, reading, comprehension and matching. In this research, annotation is the process of identifying the concept units in the corresponding semantic templates that can be matched to the designated vulnerability information pieces, so that the software fault, weakness, resource and consequence, even the fix pattern, can be recognized and given a tag for this vulnerability.

To reduce the manual labor required for the vulnerability understanding, analyzing, identification and prediction, automatic or semi-automatic assistance is required. To annotate a specified vulnerability, first we need to know the category it belongs to, which indicates the semantic template that will be used as its concept structure. So as a prerequisite of the semi-automatic annotation effort, one goal of this research is to identify certain categories (Buffer Overflow, Information Exposure, Injection, etc.) for the vulnerabilities reported in the vulnerability database, so that the appropriate semantic template can be selected to annotate a vulnerability report. The classification work and results will be described in section 7.1.

Section 7.2 will elaborate our efforts on semi-automating the vulnerability report annotation with the aid of a related semantic template. As the effects of natural language characteristics on the results of natural language processing and relevant technologies are unknown, we apply different methods of classification to our data set to compare their effectiveness and efficiency. We investigate three methods: Documents query as vectors; Documents match based on tf-idf values and selected classification models.

7.1 Classifying CVEs to Weakness Categories

The objective of category classification is to identify the most likely weakness category for a particular vulnerability. In this section, we present our results investigating the efficacy of a machine learning solution to category classification, taking information from CWEs whose categories are known in order to train a classifier to identify a weakness category for vulnerabilities. For simplicity, we used a binary classification approach where we classify vulnerabilities as buffer overflow or not. The buffer overflow category was chosen in this investigation to maintain continuity with the previous chapter.

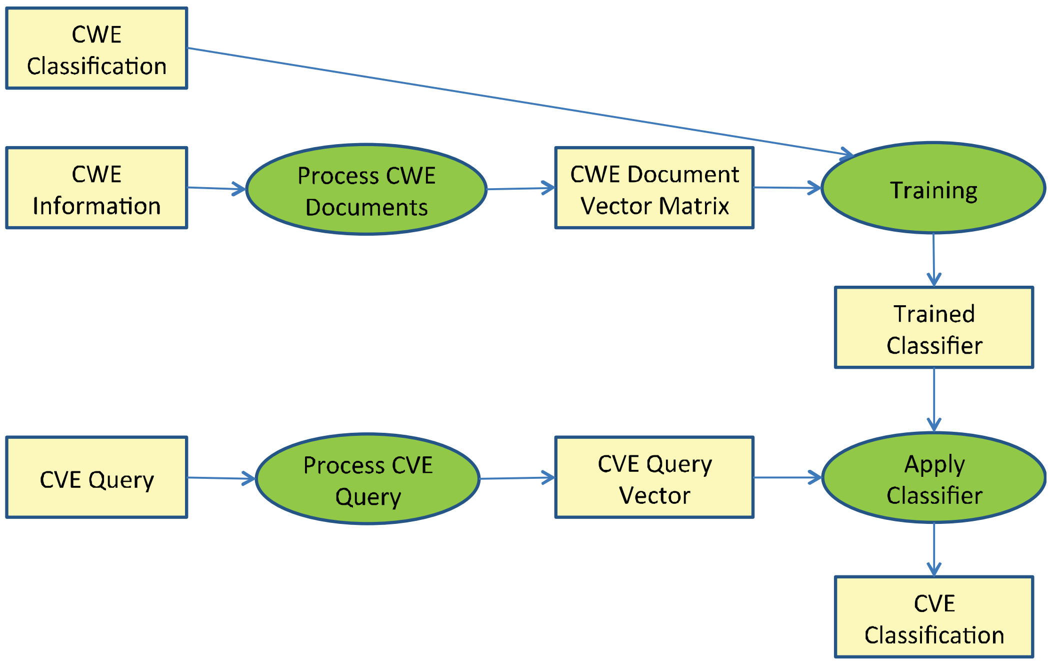


Figure 21 – Machine learning process

The machine learning approach is illustrated in Figure 21. In this dataflow diagram, we took a sample of buffer overflow-related CWEs identified in the buffer overflow semantic template and collected their associated information from the CWE (including name, summary, descriptions, consequences, examples, known CVEs, related CAPEC entries, etc.) into a comprehensive document, one document for each CWE. The documents are processed, which involves tokenization, removal of stop words, and stemming, and finally conversion into a document vector matrix, consisting of *D* rows, where *D* is the number of CWEs, and *T* columns, where *T* is the number of distinct terms in the documents. For each CWE *d*, $V_t(d)$ is a measure of frequency of term *t* in document *d*. Frequency has three commonly used measures:

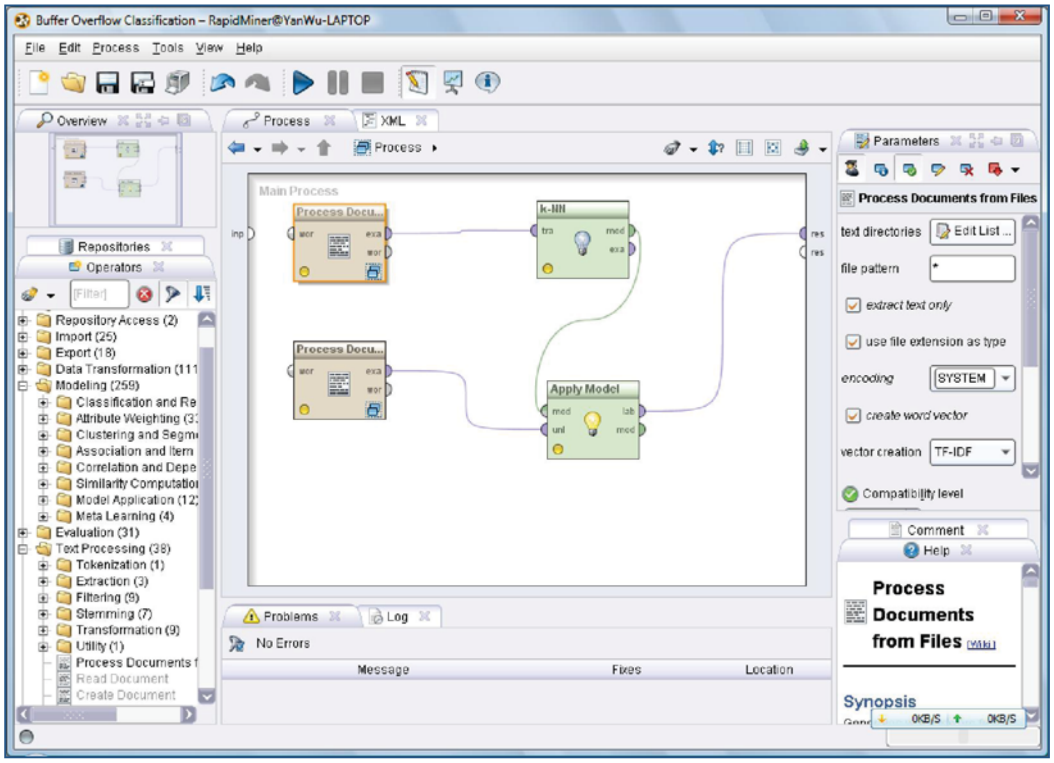
1. term occurrence – number of times a term occurred in a document
2. term frequency – term occurrence / number of terms in document
3. term frequency-inverse document frequency (tf-idf) – term frequency / number of documents the term appears in

The matrix is passed on to the trainer along with the learned answer (buffer overflow/not buffer overflow) producing a trained model. The information concerning an actual vulnerability is also transformed into another document vector which is passed to the classifier which uses the trained model to classify the vulnerability.

The number of CWE entries that were collected and represented in our Buffer Overflow Semantic Template was 44, including the software fault, weakness, resources and consequences related. So, another 44 non-Buffer Overflow CWE entries were randomly selected and their relevant natural language descriptions were collected.

We used a popular data mining tool, RapidMiner, to conduct the study.

RapidMiner was chosen because it already has implementations for a large number of classifiers.



text directories: Edit List ...
file pattern: *
extract text only: ✓
use file extension as type: ✓
encoding: SYSTEM
create word vector: ✓
vector creation: TF-IDF
Compatibility level

No Errors

Figure 22 - Classification by RapidMiner

Figure 22 shows how the classification process was realized in RapidMiner. The two “Process Documents” operators transform CWE and vulnerability information into tf-idf document vector matrices. The “Classify” operator performs the training and the “Apply” operator classifies the vulnerabilities into the most likely category.

We experimented on four simple and well-known classification algorithms (Default Model, K-Nearest Neighbor, Naïve Bayes and Naïve Bayes (Kernel)) that have been realized as operators in RapidMiner.

To verify the trained classification model, it was first applied back to the original training set. Table 14 shows that the resulting precision and recall values from the four different classification algorithms are rather high, giving confidence on this trained classification model although other concerns such as the different level of description and quality of CWE entries compared with the real world vulnerability reports might affect the application of CWE-trained classification model on the vulnerability instances.

Table 14 - Precision and Recall for Classification Model Validation

	CWE Validation	Precision	Recall
K-NN Algorithm	K = 1	100%	100%
	K = 2	84.6%	100%
	K = 3	93.5%	97.7%
	K = 4	89.8%	100%
	K = 5	91.5%	97.7%
Naïve Bayes		100%	100%
Naïve Bayes (Kernel)		97.7%	100%

After the validation of this classification model, it was applied on a set of CVE natural language descriptions (including the descriptions both from NVD and Apache Website). Similar to the CWE training set, seven Buffer Overflow CVE instances were combined with seven randomly selected non-Buffer Overflow CVE instances, and this set was classified by the trained model. The resulting precision and recall are shown as in Table 15. Except for the recall for K-NN (K=1, 2) and Naïve Bayes (Kernel), other values are relatively low, which can be partly explained by the different vocabularies used by the CWE authors versus the vulnerability recorders. As a general knowledge base for weaknesses, the CWE descriptions tend to use more abstract terms while the real world vulnerability descriptions are concerned about concrete files and faults. To improve the precision and recall, CVE instances whose categories are known can be incrementally added to the training set.

Table 15 - Precision and Recall for Classification of Buffer Overflow CVEs by the Trained Model

	CWE Validation	Precision	Recall
K-NN Algorithm	K = 1	16.9%	71.4%
	K = 2	15.3%	92.9%
	K = 3	50%	7.1%
	K = 4	22.2%	57.1%
	K = 5	20%	14.3%
Naïve Bayes		8.3%	14.3%
Naïve Bayes (Kernel)		16.9%	71.4%

7.2 Classifying CVEs to Semantic Template Concepts

The objective of concept classification is, given that we know which weakness category a vulnerability falls into, to identify the most applicable semantic template concepts within that category. The aim is to map vulnerabilities to at least one concept in each of the four aspects (software faults, weakness, resource/location, consequences) of a semantic template. This provides a semi-automated process for annotating an occurrence of vulnerability by associating it with corresponding concepts in the semantic template, which in turn would facilitate learning of the vulnerability.

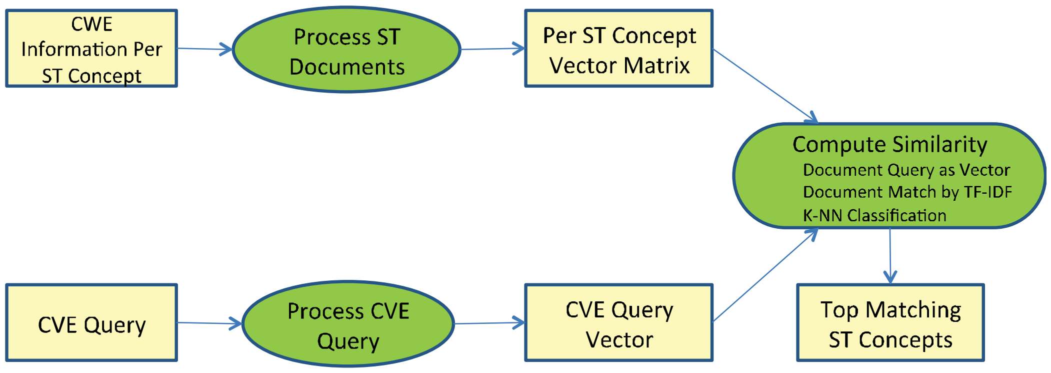


Figure 23 – Concept classification process

The concept classification techniques studied here use several variations of the process depicted in Figure 23. We employ three techniques which differ in how the similarity is computed:

1. Document query as vector – similarity is computed as the cosine similarity between pairs of concept and CVE document vectors
2. Document match based on tf-idf – similarity is computed by summing up the applicable tf-idf measures in the CWE document vector matrix
3. Classification – similarity is computed based on the probability values calculated in the process of performing machine learning and classification;

Due to the limited scope of this dissertation research work, and to keep the simplicity of narration, only the Buffer Overflow "Software Fault" component was chosen as an example to show the different semi-automatic annotation technologies. This component includes 21 concept nodes which totally contain 29 CWE entries. The test set includes six previously annotated buffer overflow CVE entries (expert manually annotated buffer overflow CVE instances, which were used in the experiment in Chapter 6), eight other buffer overflow CVEs, six cross-site scripting CVEs and one information leak CVE, explicitly identified by CVE authors as such. To keep the most meaningful contents of each CWE entry when annotating, other than the title, summary, description, we also collected the alternate terms, common consequences, demonstrated examples (source code deleted), observed examples (CVEs that have been identified to be relevant to this CWE entry, with only title and descriptions) and the related attack patterns (CAPEC id and titles). Each CWE node in software fault component may contain more than one CWE entry, and in this case the content of those CWE entries will be combined in one text document. The CVE information contains the CVE descriptions from both the Apache website and the NVD summary.

As the benchmark of this exploratory work, the experts' annotation results are as follows:

Vulnerability	**Software Fault Concepts in Semantic Template**
CVE-2004-0492	Improper-Input-Validation and Improper Handling of Length Parameter Inconsistency;
CVE-2004-0493	Improper Validation of array Index, Incorrect-Buffer-Size-Calculation, Sign Errors and Incorrect-Calculation;
CVE-2005-1268	Off-by-One and Incorrect-Calculation;
CVE-2009-0023	Improper-Input-Validation, Wrap-Around Error, Integer Underflow, Integer Coercion Error, Sign Errors and Incorrect-Calculation;
CVE-2009-2412	Improper-Input-Validation, Wrap-Around Error, Integer Overflow and Incorrect-Calculation;
CVE-2010-0010	Wrap-Around Error, Integer Coercion Error, Integer Overflow and Incorrect-Calculation.

7.2.1 Documents Query as Vectors

As one of the basic algorithms for search engine to query among large quantity of documents based on users' query terms, the document query as vectors method treats documents as a vector space and each document as a vector. The representation of a set of documents as vectors in a common vector space is known as the vector space model and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering [24].

In the document query as vectors technology [41], each document will be represented by a vector $\vec{V}$(d), with one component in the vector for each dictionary term. Unless otherwise specified, it is assumed that the components are computed by the tf-idf weighting scheme, to represent the weight of each term in this specific document set. But based on the fact that our annotation work is a dynamic job which can include more documents in the future, the term occurrence was utilized to evaluate the weight of each term, so that the effect of document set number and quality could be ignored. The set of documents in a collection then may be viewed as a set of vectors in a multi-dimension vector space, in which there is one axis for each term. This representation weighs the importance of each term with losing the relative ordering of the terms in each document.

How do we quantify the similarity between two documents in this vector space? A first attempt might consider the magnitude of the vector difference between two document vectors. This measure suffers from a drawback: two documents with very similar content can have a significant vector difference simply because one is much longer than the other. Thus the relative distributions of terms may be identical in the two documents, but the absolute term frequencies of one may be far larger. To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d_1 and d_2 is to compute the cosine similarity of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

where the numerator represents the dot product (also known as the inner product) of the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$, while the denominator is the product of their Euclidean lengths. Let $\vec{V}$ (d) denote the document vector for d, with M components $V_1(d) \cdots V_M(d)$. The dot product $\vec{V}(d_1) \cdot \vec{V}(d_2)$ of two vectors is defined as

$\sum_{i=1}^{M} V_i(d_1)V_i(d_2)$. The Euclidean length of d is defined to be $\sqrt{\sum_{j=1}^{M} V_j(d)^2}$.

The effect of the denominator of the previous equation is thus to length-normalize the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1) / |\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2) / |\vec{V}(d_2)|$. We can then rewrite the previous formula as :

$$\mathrm{sim}\,(d_1, d_2) = \overset{-->}{\upsilon}(d_1) \cdot \overset{-->}{\upsilon}(d_2)$$

As a query algorithm, the basic idea of cosine similarity is to calculating the cosine value of the angle between the query term (query document) and the objective document. So based on this similarity calculation formula, the term occurrence matrix for the natural language descriptions of 21 buffer overflow software fault CWE nodes and the corresponding CVE was calculated by Rapidminer. For each CVE, the cosine similarity between it and every CWE node concerning buffer overflow software fault was calculated and compared, and the CWE node holding the highest cosine similarity value was treated as the most possible result of this certain query. The top three CWE nodes with the highest cosine similarity value would be selected as the semi-automatic annotation results.

To validate this annotation method, all the cosine similarity values are collected and showed in Figure 24. The green bars showed the buffer overflow CVEs and the empty bars non-buffer overflow, and the cosine similarity difference between the two groups of CVEs was tested by the wilcoxon test. With the W =35875.5 and p value < 2.2e-16, it can be confirmed that the buffer overflow CVEs hold the greater similarity value with the buffer overflow CWEs compared with the non-buffer overflow CVEs.

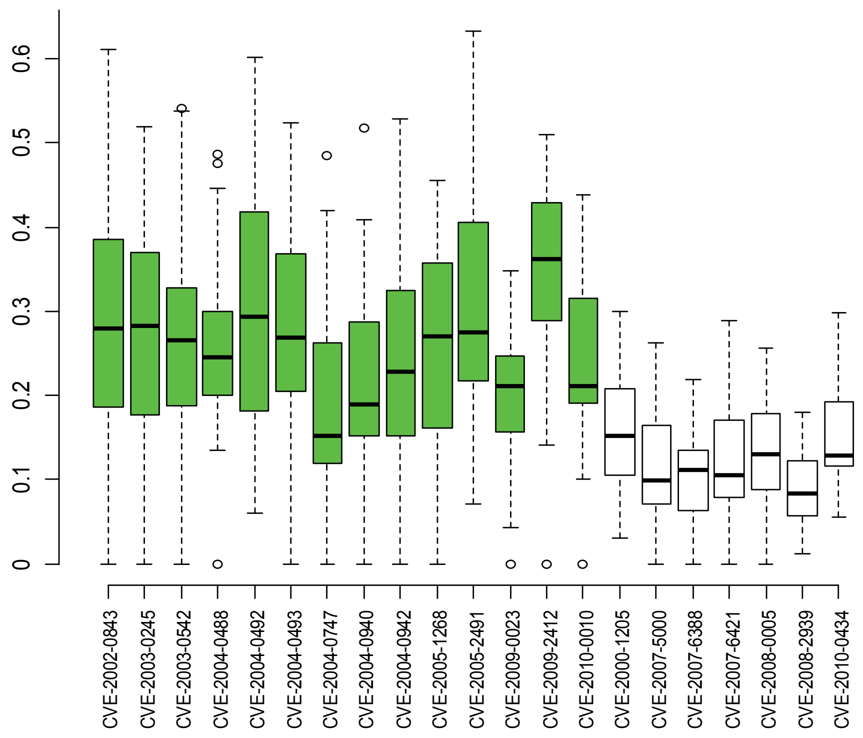


Figure 24 - Cosine Similarity of Document Vectors

7.2.2 Documents Match based on TF-IDF

Different from term occurrence, the tf-idf values evaluate the importance of certain dictionary term in a document with the consideration of the whole document set. With the calculation of term frequency divided by Inverse document frequency for certain term in certain document, its value will be higher for the terms appear more times in the specified document and seldom appear in other documents in the same document set, so that the “uniqueness” and “representativeness” of those dictionary terms can be identified to reflect the characteristics of the document it belongs to.

In this research work, all the CWE concepts within software fault component and each object CVE description will be fed into Rapidminer, by the operators: “Process Documents from Files” (“Tokenize”, “Filter Stopwords” then “stem”) and “Write csv”. The only difference is that the CWE document set will be analyzed by tf-idf vector creation method but the CVE document will be analyzed by the binary occurrence. The output are two .csv document with one tf-idf matrix for this CWE document set, in which each line represents one document, each column represents one dictionary term appeared in this document set and the cross-grid represents the tf-idf value for each term in the corresponding document; and another binary term occurrence matrix for the CVE document. A java program was designed by the author to separately read the CWE tf-idf matrix and the CVE binary term occurrence matrix, then compare the two matrixes, when the same dictionary term with a non-zero value appears in both of the CVE and CWE document, it would be recorded with the corresponding tf-idf value in the CWE document. The Sum of all the tf-idf value in one CWE document that matched with the CVE would be treated as the indication of how well the two documents are matched. Then the top five matched CWE entries were selected and recorded.

To validate this annotation method, all the tf-idf similarity values are collected and showed in Figure 25. The green bars showed the buffer overflow CVEs and the empty bars non-buffer overflow, and the tf-idf similarity difference between the two groups of CVEs was tested by the wilcoxon test. With the W =24787.5 and p value = 0.005885, it can be confirmed that the buffer overflow CVEs hold the greater tf-idf similarity value with the buffer overflow CWEs compared with the non-buffer overflow CVEs.

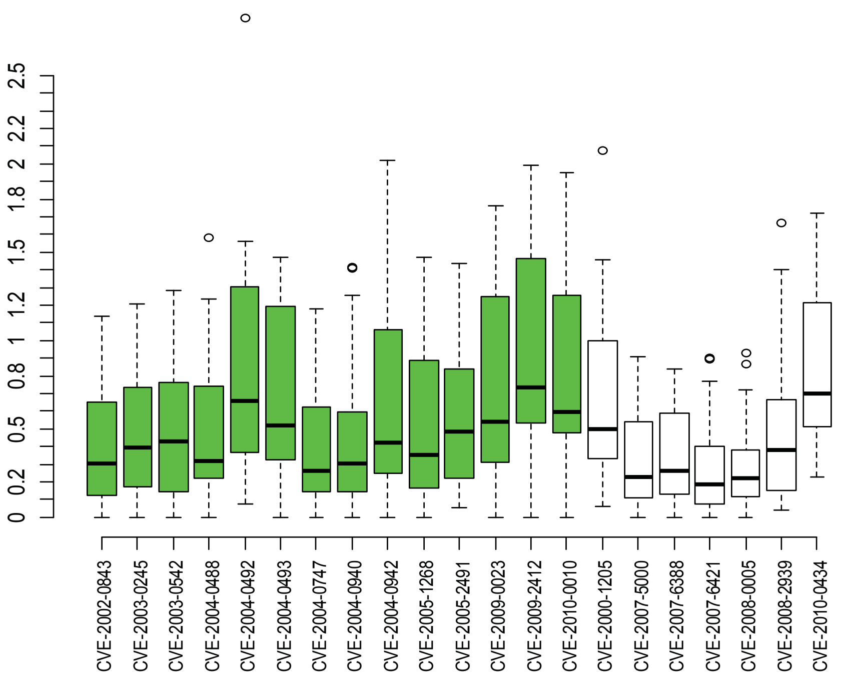


Figure 25 - tf-idf Similarity

7.2.3 K-NN Classification of CVEs to Concepts

Rapidminer, as an open source system for data mining, incorporated most of the known data mining algorithms and wrapped them into individual operators which can be flexibly combined to reach specific goals for researchers and industry practitioners. Classification, as a typical data mining task, was implemented by Rapidminer as a set of modeling operators. Also to keep the simplicity and the scope of this research in consideration, only the K-nearest neighbor (K=5) algorithm with four different measures was selected, they are: Euclidean Distance, Cosine Similarity, Jaccard

Similarity and the Overlap Similarity. The first two are numerical similarity measures while the last two are set-based measures.

The reason for selecting the four measures is to obtain certain diversity so that the results from different measures could be representative. The Euclidean distance is the "ordinary" distance between two points that one would measure with a ruler. By using this formula as distance, Euclidean space (or even any inner product space) becomes a metric space. Cosine similarity is a measure of similarity between two vectors by measuring the cosine of the angle between them. The result of the Cosine function is equal to 1 when the angle is 0, and it is less than 1 when the angle is of any other value. Calculating the cosine of the angle between two vectors thus determines whether two vectors are pointing in roughly the same direction. The Jaccard index, also known as the Jaccard similarity coefficient (Wikipedia), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

The overlap coefficient is a similarity measure related to the Jaccard index that computes the overlap between two sets which is defined as follows:

$$\mathrm{overlap}(X,Y) = \frac{|X \cap Y|}{\min(|X|,|Y|)}$$

If set X is a subset of Y or the converse then the overlap coefficient is equal to 1.

The four K-NN algorithms worked on the same set of data and produced different annotation results based on different calculation and different data type.

7.3 Concept Classification Results

The top five classification results for each technology are listed in Table 16. By comparing with the expert annotation results, all the matched CWE nodes were highlighted by underline. By observation, most of them contain one or two overlaps with the expert annotation results, and among the different algorithms, the CWE tf-idf CVE binary algorithm shared the most overlaps with the query as vector algorithm on their top five results for all the CVE annotations. The Jaccard and Overlap measurements also share around 60% of their top five results, which is not surprising based on the fact that both of them depend on the set calculation.

Table 16 – Top Five Results of Annotation Methods

	2004-0492	2004-0493	2005-1268	2009-0023	2009-2412	2010-0010
Documents Match based on TF-IDF	***130***	130	***193***	130	***190-680***	130
	131	***131***	130	131	130	***190-680***
	120	193	170	193	***20***	131
	193	190-680	131	***20***	193	193
	190-680	***129-789***	120	190-680	120	120
Precision	16.7%	42.9%	20%	16.7%	50%	33.3%
Recall	50%	42.9%	50%	12.5%	60%	40%
Query as Vectors	***130***	***129-789***	***193***	131	***190-680***	131
	131	193	131	130	131	130
	120	130	120	***20***	120	***190-680***
	193	20	190-680	193	130	120
	190-680	***131***	130	129-789	193	193
Precision	16.7%	50%	16.7%	16.7%	33.3%	33.3%
Recall	50%	42.9%	50%	12.5%	40%	40%

Euclidean	***130***	***129-789***	227	227	227	194-195-196
	415-416	***194-195-196***	192	415-416	415-416	***192***
	20	415-416	415-416	242	242	415-416
	194-195-196	192	242	***20***	192	242
	227	227	194-195-196	***192***	194-195-196	227
Cosine	***130***	***129-789***	***193***	***20***	***190-680***	194-195-196
	129-789	20	170	190-680	***20***	130
	190-680	***194-195-196***	190-680	130	227	***190-680***
	20	193	227	***191***	129-789	120
	120	190-680	129-789	227	120	467-468
Jaccard	193	120	***682***	131	***20***	120
	120	190-680	191	***20***	242	467-468
	129-789	***194-195-196***	190-680	130	129-789	***190-680***
	190-680	193	***193***	190-680	456	130
	130	***129-789***	170	***191***	***190-680***	194-195-196
Overlap	191	193	170	227	242	***190-680***
	190-680	128	131	131	129-789	467-468
	194-195-196	242	120	***20***	120	242
	129-789	***129-789***	190-680	130	***20***	194-195-196
	130	***194-195-196***	***193***	190-680	***190-680***	***192***

There are overlaps and differences among all the similarity calculation among the four K-Nearest Neighbor annotation algorithms, so a weighted voting is executed to select the top five ST concepts that could be representative for K-NN to annotate the corresponding CVEs. Basically, for the results from the four different algorithms, the first place voting are given a weight as 1, second as 0.8, third as 0.6, fourth as 0.4 and the fifth as 0.2. And to mitigate the bias, the ST concepts that appear only once are rejected. Table 17 shows the voted ranking results (keep the top five) for the four K-NN algorithms.

Table 17 – Voting Results for the Four K-NN Annotation Algorithms

	2004-0492	2004-0493	2005-1268	2009-0023	2009-2412	2010-0010
1	***<u>130</u>***/ 129-789/ 190-680	***<u>129-789</u>***	227	131	227/ 242	***<u>190-680</u>***
2	***<u>20</u>***/ 194-195 -196/ 120	193	170	227	***<u>20</u>***	120
3		194-195 -196	***<u>193</u>***/ 190-680	***<u>20</u>***	129-789	194-195 -196
4		190-680		130	***<u>190-680</u>***	130/ 467-468
5				190-680	120	***<u>192</u>***/ 242
Precision	20%	25%	20%	16.7%	37.5%	27.3%
Recall	100%	28.6%	50%	12.5%	60%	60%

Then, with the expert annotation as benchmark, precision and recall values for all the results from the three algorithms (CWE tf-idf CVE Binary, Query as Vector, K-NN) were calculated and shown in Table 18. From the voting results, the observation is the recall is relatively high.

Table 18 – Voting Results for all the three Annotation Algorithms (Documents Match based on TF-IDF, Query as Vectors and K-NN)

	2004-0492	2004-0493	2005-1268	2009-0023	2009-2412	2010-0010
1	***130***	130	***193***→***682***	131→***682***	***190-680*** →***128***	***190-680*** →***128***/131→***682***
2	131→682	193→***682*** /***129-789***	170	130	***20***	130
3	120	***131***	131	***20***	130	120
4	190-680 →128	190-680 →128	130	193	120	193
5	193		190-680 →128	190-680 →***128***	193→***682***	
Precision	12.5%	44.4%	25%	37.5%	62.5%	50%
Recall	50%	57.1%	100%	37.5%	100%	80%

The weighted voting can reflect the overall ranking among the different annotation algorithms but the cover-all approach and the average calculation might lose certain important ranking information, to keep the original ranking information, we used another method named max positioning to summarize the top five results, which first collect all the first placed nodes from all the four annotation algorithms, delete the repeated ones, put in the first place of results, then repeat this process on the second, to fifth places, and delete the nodes if they appeared in the previous places, the final results are shown in Table 19. To keep the completeness, the root nodes are inserted into the results other than the original results.

Table 19 – Max Positioning Results

	2004-0492	2004-0493	2005-1268	2009-0023	2009-2412	2010-0010
1	***130/*** 129-789/ 190-680→128 →682	130/ ***129-789***	***193***→***682/*** 227	130/ 131→***682***	***190-680***→ ***128-682/*** 227/ 242	130/ 131→***682/*** ***190-680***→ ***128***
2	131/ ***20/*** 194-195 -196/ 120	***131***→***682***/193	130/ 131/ 170	227	130/ 131/ ***20***	120
3		***194-195-196***	120/ 190-680→ 128	193/ 20	120/ 129-789	194-195-196
4	193	190-680→128/ 20		190-680→ ***128***/ 129-789	193	193/ 467-468
5						***192/*** 242
Precision	14%	53.8%	20%	18.2%	38.5%	33.3%
Recall	100%	100%	100%	25%	100%	100%
Only Top Results						
Precision	14%	66.7%	66.7%	33.3%	66.7%	66.7%
Recall	50%	28.6%	100%	12.5%	80%	80%

Based on the precision and recall calculation, we conducted a pairwise comparison of the performance of the three annotation approaches using paired, one-tailed Wilcoxon tests. The results are as follows:

Precision:

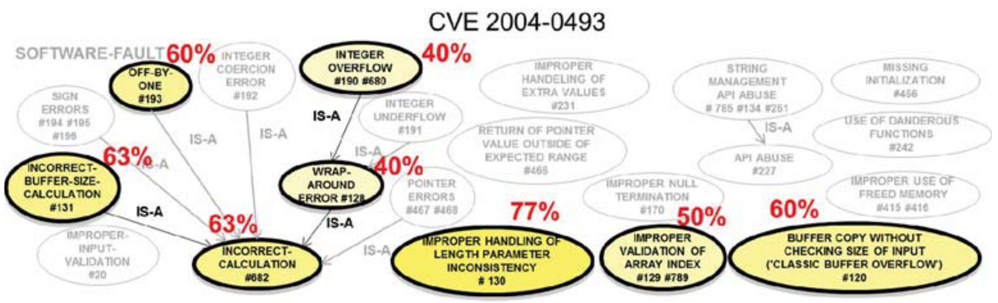
- There are no statistically significant differences between the precision of K-NN, TF-IDF and cosine similarity.
- Max positioning is statistically better than K-NN, TF-IDF or cosine similarity.
- Max positioning is statistically better than voting.

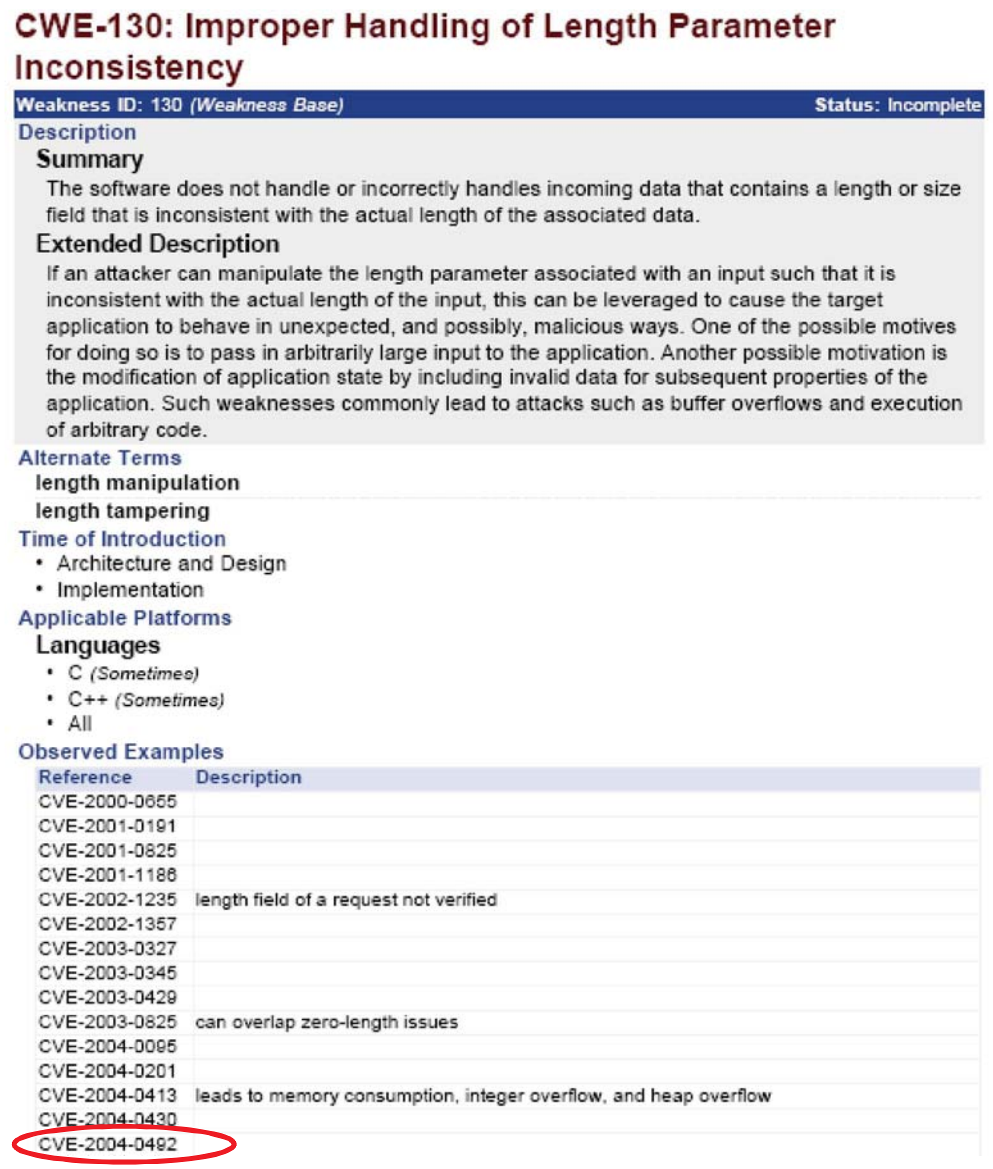
Recall:

- There are no statistically significant differences between the recall of K-NN, TF-IDF and cosine similarity.
- Voting is statistically better than TF-IDF or cosine similarity but not statistically different from K-NN
- Max positioning is not statistically different from K-NN, TF-IDF or cosine similarity
- Voting is statistically better than max positioning

In this chapter we described the integration of multiple machine learning techniques to develop features (distinguishing keywords in CWE descriptions) for each semantic template concept. These results are then used for the semi-automatic annotation of natural language vulnerability descriptions in software repositories.

To further summarize and evaluate the annotation results, the top five results from tf-idf, documents as vector from Table 16, and the K-NN voting Table 17 for CVE-2004-0492 are gathered and voted with the same weighted voting rule as used for Table 17. For this time, the weighted ranking results are listed as percentages for each selected node in Figure 26, and the root nodes inherit the maximum ranking results from all of their children. As the weighted voting results, this visualization reflects the probabilities for each selected semantic template node to be the "correct tag" to annotate CVE-2004-0492. The second half of Figure 26 was a segment of description from CWE specification 1.6, for CWE-130, which listed CVE-2004-0492 (marked by red circle) as one of the observed examples, and also CWE-130 was in the expert identified relevant CWE entries for CVE-2004-0492. This visualization enhanced our confidence on this set of semi-automatic annotation techniques.


CVE 2004-0493
SOFTWARE-FAULT
60%
OFF-BY-ONE #193
INTEGER COERCION ERROR #192
INTEGER OVERFLOW #190 #680
40%
SIGN ERRORS #194 #195 #196
IS-A
IS-A
IS-A
INTEGER UNDERFLOW #191
IMPROPER HANDELING OF EXTRA VALUES #231
STRING MANAGEMENT API ABUSE # 785 #134 #251
MISSING INITIALIZATION #456
RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE #465
IS-A
USE OF DANDEROUS FUNCTIONS #242
63%
IS-A
INCORRECT-BUFFER-SIZE-CALCULATION #131
IS-A
WRAP-AROUND ERROR #128
40%
API ABUSE #227
POINTER ERRORS #467 #468
IMPROPER NULL TERMINATION #170
IMPROPER USE OF FREED MEMORY #415 #416
IS-A
63%
IS-A
77%
50%
60%
IMPROPER-INPUT-VALIDATION #20
INCORRECT-CALCULATION #682
IS-A
IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY # 130
IMPROPER VALIDATION OF ARRAY INDEX #129 #789
BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW') #120



CWE-130: Improper Handling of Length Parameter Inconsistency

Weakness ID: 130 *(Weakness Base)* **Status: Incomplete**

Description

Summary

The software does not handle or incorrectly handles incoming data that contains a length or size field that is inconsistent with the actual length of the associated data.

Extended Description

If an attacker can manipulate the length parameter associated with an input such that it is inconsistent with the actual length of the input, this can be leveraged to cause the target application to behave in unexpected, and possibly, malicious ways. One of the possible motives for doing so is to pass in arbitrarily large input to the application. Another possible motivation is the modification of application state by including invalid data for subsequent properties of the application. Such weaknesses commonly lead to attacks such as buffer overflows and execution of arbitrary code.

Alternate Terms

length manipulation

length tampering

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C *(Sometimes)*
- C++ *(Sometimes)*
- All

Observed Examples

Reference	Description
CVE-2000-0655	
CVE-2001-0191	
CVE-2001-0825	
CVE-2001-1186	
CVE-2002-1235	length field of a request not verified
CVE-2002-1357	
CVE-2003-0327	
CVE-2003-0345	
CVE-2003-0429	
CVE-2003-0825	can overlap zero-length issues
CVE-2004-0095	
CVE-2004-0201	
CVE-2004-0413	leads to memory consumption, integer overflow, and heap overflow
CVE-2004-0430	
CVE-2004-0492	

Figure 26 - Weighted Voting Result for CVE-2004-0492

8. Conclusion

The goal of this work was to develop a conceptual framework for the study of software vulnerabilities, and examine its effectiveness in practice for use by stakeholders in the SDLC. To this end, we have outlined a process for the construction of semantic templates, annotation of vulnerability artifacts using the semantic templates manually as well as semi-automatically, and evaluating the effectiveness of semantic templates through a controlled experiment. Vulnerability artifacts and trends from a large-scale open source software development project provided units of analysis throughout this work. Our experiences demonstrate feasibility of constructing semantic templates for various weakness types from existing body of knowledge for software weaknesses. Our findings indicate that semantic templates do improve the effectiveness of novice programmers to study software vulnerabilities, in addition to providing semantic guidance to machine learning techniques used for semi-automatically annotating natural language vulnerability descriptions in software repositories. Our results are also geared towards improving the adoption of software assurance standards, thus, the contributions of this work are far-reaching and of immediate relevance to a body of security/software engineering researchers and practitioners.

As an organized conceptual model to study and avoid software vulnerabilities, semantic templates hold the potential application for improving current education and training practices in software development. With the observation of the improvement of students' performance in the experiment, semantic templates help novice developers to spend less time on understanding vulnerabilities, while achieving higher recall on identification of relevant weakness concepts. Also, semantic templates complement current knowledge bases for weaknesses such as the CWE and CVE.

Software repositories are rich sources of information about vulnerabilities that occur during a product's lifecycle. Although available, such information is scattered across numerous databases. Furthermore, in large software repositories, a single vulnerability may span across multiple components and have multidimensional interactions with other vulnerabilities. Thus, identifying the patterns of vulnerability occurrence in a larger context of software development continues to be an open problem. Semantic templates embody and integrate scattered patterns of vulnerability occurrences and provide a handle on the information overload problem that developers face during the study of software vulnerabilities. The organized structure of semantic templates offers a repeatable and intuitive schema for the representation and management of vulnerability information. Such information provide actionable metrics and measures to guide programmer training, authoring descriptions and organization of standard weakness enumerations, automated tool development, secure coding guidance, as well as allocation of resources towards secure software development efforts.

8.1 Contributions

As described in this dissertation, contributions of this research work include:

- Developed a process for constructing semantic templates to study software vulnerabilities by analyzing and aggregating CWE weaknesses.

    - Demonstrated the use of this process to design the Buffer Overflow, Injection, and Information Exposure semantic templates
    - Implemented a representation of semantic templates using the Web Ontology Language (OWL) as well as motivated the development of queries based on concepts in the semantic template to retrieve corresponding vulnerability information recorded in a semantic web repository.

- Demonstrated the use of the semantic templates by manually annotating publicly reported software vulnerabilities from a large project. This effort improved the understanding of the vulnerabilities as well helped refine/verify the template structure.
- Empirically validated the efficiency and effectiveness of semantic templates using a controlled experiment. In addition, expert surveys indicate wide applicability and usefulness of semantic templates for other research and development efforts for software vulnerabilities.
- Integrated multiple machine learning techniques to develop features (distinguishing keywords in CWE descriptions) for each semantic template concept, which is used for semi-automatic annotation of natural language vulnerability descriptions in large software repositories.

Table 20 maps the research activities described in this dissertation to the study hypotheses.

Table 20 - Mapping between Research activities and the Hypotheses

	Study Hypotheses						
Research activities	1.a.i. Improved annotation & integration	1.a.ii. Improved presentation & navigation	1.b.i. Enable categorization & cataloguing of fixes	1.b.ii. Retrieve fixes from related vulnerabilities	1.c.i. Facilitate development of metrics	2.a. Improved standards review and enumerations.	2.b. Reduce inconsistency or redundancy.
5. Semantic infrastructure	+	+	+	+	+	+	++
6. Empirical Validation	+	++				++	
7. Semi-automatic Annotation	++	+			++	+	+

9. Ongoing and Future Research

The work presented in this dissertation represents initial steps towards a larger research plan. We outline future work making use of semantic templates.

9.1 Incorporating Social Network Data

Semantic templates provide a framework for organizing vulnerability-related information from software project repositories and vulnerability categorizations and databases. The analysis of project repository information has thus far been limited to change history data. Incorporating additional data gleaned from the project repository can enable more sophisticated analysis and queries.

Social network data refers to individual and relationship information of the project stakeholders of the SDLC. This includes user profiles and data gathered from recorded discussions and conversations. The Apache project repository includes a richer set of data including discussions in development mailing lists and bug databases. This information can provide additional insights into understanding how the vulnerability was discovered. Certain patterns of vulnerability appearance can be identified based on Social network analysis. For instance, a reported bug may not be immediately recognized as a vulnerability issue until further discussions with experienced developers. Additional insights can also be gained in assessing the quality of fixes. For example, a fix may set off a long thread of discussions regarding its merits and limitations, and may itself be further patched, indicating incompleteness.

9.2 Identifying and Cataloguing Fix Patterns

Relevant information about fixes to vulnerabilities will be collected, analyzed and grouped based on vulnerability categorization. In this process, abstract patterns of similar fixes will be extracted from the commonalities among the source code changes associated with the fix using the following proposed process:

- Manually inspect description and source code changes of fixes to identify the core intents and activities of them
- Generalize and develop categories of the identified fixes
- Investigate if there are relationships between fix categories and change logs (do certain keywords show up?)
- Investigate similarities and relationships between vulnerabilities categories and fix categories (do certain vulnerabilities get fixed a certain way?)

9.3 Ontology Reasoning on Vulnerabilities

We have implemented semantic templates by constructing a rudimentary ontology of software weaknesses. The semantic templates were manually constructed and we have shown how instances of vulnerabilities can be classified through manual or semi-automatic annotation. The ontology's schema currently consists of basic taxonomical relationships between semantic template concepts. Additional properties (e.g. possible fix pattern for a certain category of weakness), axioms, constraints can be added in order to construct a working knowledge base. The future work on ontology development would be collecting such additional properties, constraints to complete the weakness knowledge base so that more advanced functionalities such as reasoning and inference would be feasible.

- Based on the source code comparison before and after fixes, develop fix categorizations and report their frequencies with respect to different weaknesses;
- Social network analysis to look for the patterns of vulnerability appearance;
- By reasoning in the vulnerability knowledge base, a developer could link to information such as “other possible vulnerabilities” or “possible solutions”

9.4 Semantic Templates for Education and Training

Although weaknesses are inevitable in software systems, an experienced developer or testing engineering generally could discover possible vulnerabilities faster and more accurate than novice counterparts. Developing this ability to sense an occurrence of vulnerability is generally nurtured by time and experience.

Semantic templates were designed to help developers easily follow the lifecycle of certain categories of vulnerabilities, including the origin (software faults), appearance (weakness), location (resource) and the results (consequences). With the additional CAPEC and fix patterns information, this knowledge structure can be the basis for teaching materials for developers, especially novices, to learn the nature and solutions of vulnerabilities.

9.5 Domain Specific Semantic Templates

Current semantic templates are based on weakness categories (such as buffer overflow, injection etc.), which is how we divide and conquer the vast number of weaknesses in the CWE. Using other perspectives, the scope of existing semantic templates can be further tailored. Several semantic templates representing different categories of weaknesses could share certain concept units. For example, improper input validation (CWE-20) is the shared software fault for both buffer overflow and injection. Also, as a consequence-based category of weaknesses, the denial of service semantic template that we planned is expected to share several consequence concept units with other semantic template. Based on this situation, a more abstract or specific collection of semantic template concepts could be used to organize a group of other relevant semantic templates.

While from a domain specific perspective, even current categories of weaknesses, such as injection, which span several domains and technology groups, may contain irrelevant concepts that add noise to the mental model of developers. Domain specific view helps us tailor existing semantic templates to improve developer understanding of vulnerabilities. Common Weakness Scoring System (CWSS), as a common framework for prioritizing weaknesses that are discovered in software applications, in conjunction with the Common Weakness Risk Analysis Framework (CWRAF), can be used by consumers to identify the most important weaknesses for their business domains, in order to inform their acquisition and protection activities as one part of the larger process of achieving software assurance. Figure 27 shows the business domains and technology groups identified as part of the CWRAF. Domain specific semantic templates may be tailored from the CWEs at the intersection of business domains and technology groups (referred to as vignettes) in Figure 27.

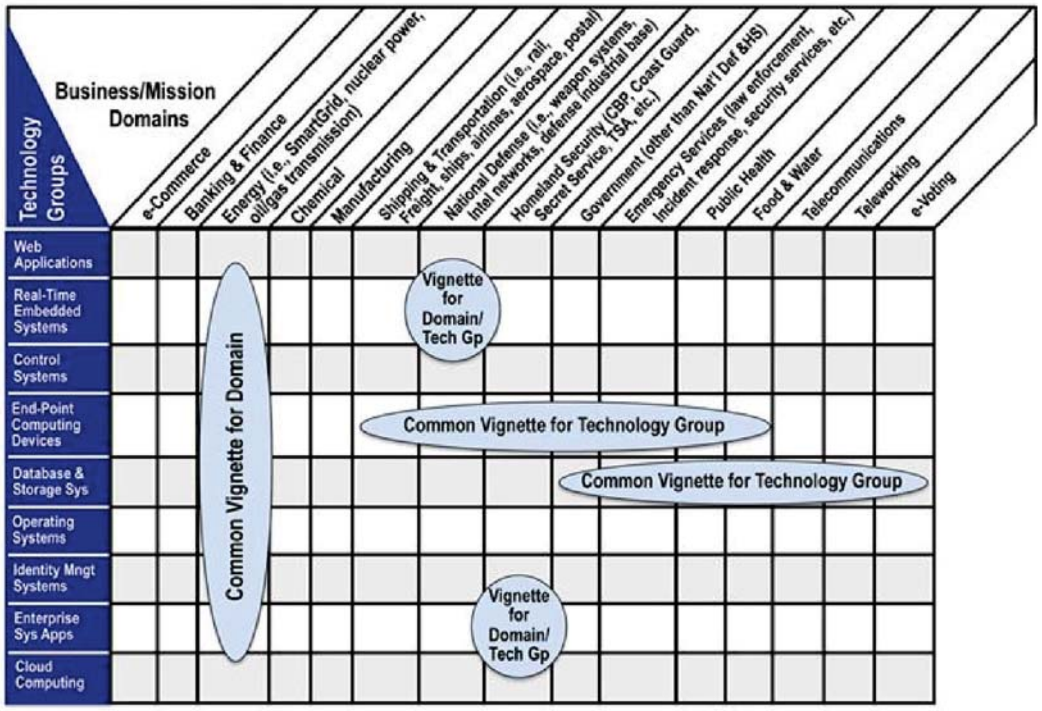

Business/Mission Domains
Technology Groups
e-Commerce
Banking & Finance
Energy (i.e., SmartGrid, nuclear power, oil/gas transmission)
Chemical
Manufacturing
Shipping & Transportation (i.e., rail, Freight, ships, airlines, aerospace, postal)
National Defense (i.e., weapon systems, intel networks, defense industrial base)
Homeland Security (CBP, Coast Guard, Secret Service, TSA, etc.)
Government (other than Nat'l Def &HS)
Emergency Services (law enforcement, incident response, security services, etc.)
Public Health
Food & Water
Telecommunications
Teleworking
e-Voting
Web Applications
Real-Time Embedded Systems
Control Systems
End-Point Computing Devices
Database & Storage Sys
Operating Systems
Identity Mngt Systems
Enterprise Sys Apps
Cloud Computing
Common Vignette for Domain
Vignette for Domain/ Tech Gp
Common Vignette for Technology Group
Common Vignette for Technology Group
Vignette for Domain/ Tech Gp

Figure 27 – Technology Groups and Business Domains

10. References

[1] R. P. Abbott, J. S. Chin, J.. E. Donnelley, W. L. Konigsford, S. Tokubo, and D. A. Webb. The RISOS Project: Security Analysis and Enhancements of Computer Operating Systems. *Lawrence Livermore Laboratory TR NBSIR-76-1041*, April 1976.

[2] T. Aslam. A Taxonomy of Security Faults in the UNIX Operating System. Purdue University, August 1995.

[3] K. Y. Begel, Phang and T. Zimmermann. Codebook: Discovering and Exploiting Relationships in Software Repositories. *ICSE '10*, May 2-8 2010, Cape Town, South Africa

[4] J. Bevan, E. J. Whitehead, S. Kim, M. Godfrey. Facilitating Software Evolution Research with Kenyon. *Proc. of 13th Foundations of Software Eng*. Sept. 2005.

[5] R. Bisbey and D. Hollingworth. Protection Analysis: Final Report. Information Sciences Institute, University of Southern California, ARPA ORDER NO. 2223, ISI/SR-78-13 May 1978.

[6] M. Bishop. A Taxonomy of UNIX System and Network Vulnerabilities. Department of Computer Science University of California at Davis, *CSE-95-10*, May 1995.

[7] S. M. Christey. The Preliminary List of Vulnerability Examples for Researchers (PLOVER). NIST Workshop Defining the State of the Art of Software Security Tools, Gaithersburg, MD, August 2005.

[8] S. M. Christey, C. O. Harris, J. E. Kenderdine, B. Miles, and R. Martin. CWE Version 1.6. CWE - Common Weakness Enumeration. 29 Oct. 2009. The MITRE Corporation. <http://cwe.mitre.org/>.

[9] D. W. Embley, Y. Ding, S. W. Liddle and M. Vickers. Automatic Creation and Simplified Querying Of Semantic Web Content. *In Proceedings of First Asian Semantic Conference (ASWC)*, Beijing China (2006).

[10] R. Feldman, and I. Dagan,. Knowledge discovery in textual databases (kdt). *In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 112–117. AAAI Press, 1995.

[11] R.A. Gandhi, H. Siy, Y. Wu, Studying Security Vulnerabilities, CrossTalk, *The Journal of Defense Software Engineering*, Sept/Oct issue 2010.

[12] J. H. Gennari, A. M. Mark, et al. The Evolution of Protégé-2000: An Environment for Knowledge-based Systems Development. *Proc. of Human-Computer Studies*. 1st ed. Vol. 58. 2003. Print.

[13] T. Gruber. A Translation Approach to Portable Ontologies. *Knowledge Acquisition* 5, 2, 199-299, 1993.

[14] J. Han, M. Kamber. Data Mining: Concepts and Techniques. San Francisco, CA: Morgan Kaufmann, 2006. Print.

[15] E. Hassan, R. C. Holt. The small world of software reverse engineering, *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04).*

[16] A. Hotho, A. N¨urnberger, and G. Paass. A brief survey of text mining. *LDV Forum*, 20(1):19-62, 2005.

[17] M. Howard, D. LeBlanc, J. Viega. 19 Deadly Sins of Software Security Programming Flaws and How to Fix Them. McGraw-Hill Osborne Media. ISBN: 0-07-226085-8, July 2005.

[18] E. John, E. Gansner, E. Koutsofios, S. North and G. Woodhull. Graphviz and Dynagraph - Static and Dynamic Drawing Tools. *Graph Drawing Software* (2003): 127-48. Print.

[19] C. M. Judd, R. S. Eliot, and H. K. Louise. *Research Methods in Social Relations*. Fort Worth: Holt, Rinehart, and Winston, 1991. Print.

[20] M. Kantardzic. *Data Mining: Concepts, Models, Methods and Algorithms.* New York: Wiley-Interscience, 2002. Print.

[21] C. Kiefer, A. Bernstein, J. Tappolet. Mining software repositories using iSPARQL and a software evolution ontology. *4th Int'l Workshop on Mining Soft*. 2007.

[22] S. Kim, et al. TA-RE: an Exchange Language for Mining Software Repositories. *3rd Int'l Workshop on Mining Soft*. 2006.

[23] S. Kim, K. Pan and E. J. Whitehead. Memories of bug Fixes. *SIGSOFT'06/FSE-14*, November 5–11, 2006, Portland, Oregon, USA.

[24]C. E. Landwehr, A. R. Bull, J. P. Mcdermott, AND W. S. Choi. A Taxonomy of Computer Program Security Flaws with Examples. Information Technology Division, Code 5542, Naval Research Laboratory, Washington, D.C. 20375-5337 in ACM Computing Surveys 26, 3 (Sept., 1994)

[25] D. Manning, P. Raghavan and H. Schütze. *Introduction to Information Retrieval*. New York: Cambridge UP, 2008. Print.

[26] Q. Mei, C. Zhai. Discovering evolutionary theme patterns from text – an exploration of temporal text mining, *KDD'05*, 21-24 August 2005, Chicago, Illinois, USA.

[27] K. Pan, S. Kim, E. J. Whitehead. "Toward an understanding of bug fix patterns." *Empirical Software Engineering* 14:286-315, 2009.

[28] D. Sánchez, D. Isern and M. Millan. Content annotation for the semantic web: an automatic web-based approach. *Knowledge and Information System*. DOI 10.1007/s10115-010-0302-3. 2010.

[29] Shapiro, S. S., and M. B. Wilk. "An Analysis of Variance Test for Normality (Complete Samples)." Biometrika 52.3/4 (1965): 591. Print.

[30] M. Shewhart, M. Wasson. Monitoring a newsfeed for hot topics, *Proceedings of KDD-99* San Diego CA USA, 1999.

[31] H. Siy, Y. Wu. An Ontology to Support Empirical Studies in Software Engineering. *Proceedings of the International Conference on Computing in Engineering, Science and Informatics (ICC2009)*, Fullerton, California, April 2009.

[32] S. Sudhakrishnan, J. T. Madhavan and E. J. Whitehead. Understanding Bug Fix Patterns in Verilog. *MSR'08*, 10-11 May 2008, Leipzig, Germany.

[33] J. Tang, M. Hong, D. Zhang, B. Liang, and J. Li. "Information Extraction: Methodologies and Applications." *Emerging Technologies of Text Mining: Techniques and Applications*. Hershey, PA: Information Science, 2007. Web.

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

[35] S. Weber, P. A. Karger, A. Paradkar. A Software Flaw Taxonomy: Aiming Tools at Security. *IBM Research Division, Software Engineering at Secure Systems - Building Trustworthy Applications (SESS'05)*. St. Louis, Missouri, June 2005.

[36] Y. Wu, R. A. Gandhi and H. Siy. Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories. *Proc. of The 6th International Workshop on Software Engineering for Secure Systems (SESS'10) at the 32nd International Conference on Software Engineering (ICSE 2010)*, South Africa, Cape Town. 1-8 May 2010.

[37] Y. Wu, H. Siy, L. Fan. Discovering Meaningful Clusters from Mining Software Engineering Literature. *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE '08)*, Redwood City, California, July 2008.

[38] Y. Wu, H. Siy and R.A. Gandhi, NIER: Empirical Results on the Study of Software Vulnerabilities. *NIER at the 33rd International Conference on Software Engineering (ICSE 2011)*, Honolulu, Hawaii. May 2011

[39] *CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC)*. Web. 28 Oct. 2011. <http://capec.mitre.org/>.

[40] *CVE -Common Vulnerabilities and Exposures (CVE)*. Web. 28 Oct. 2011. <http://www.cve.mitre.org>.

[41] *"Queries as Vectors."* The Stanford NLP (Natural Language Processing) Group. Web. 28 Oct. 2011. <http://nlp.stanford.edu/IR-book/html/htmledition/queries-as-vectors-1.html>.

[42] *"CWE - CWE Glossary."* CWE -Common Weakness Enumeration. Web. 28 Oct. 2011. <http://cwe.mitre.org/documents/glossary/index.html>.

[43] The Ten Most Critical Web Application Security Vulnerabilities. The Open Web Application Security Project (OWASP), January 2004.

[44] The Web Security Threat Classification. Web Application Security Consortium, November 2005.

[45] World Wide Web Consortium (W3C). *OWL Web Ontology Language reference*. Web. 28 Oct. 2011. <http://www.w3.org/TR/owl-ref/>.

11. Appendix

*A. **Glossary***

Attacker: an actor who attempts to gain access to behaviors or resources that are outside of the software's intended control sphere for that actor. [42]

Can-Precede: this is a binary relationship that indicates causal connects between two software faults, software fault and weakness, or weakness and consequence.

Consequence: failure conditions that violate security properties. [42]

Occurs-In: this is a binary relationship that indicates the presents of a weakness in a context of the resource.

Precision: In the field of information retrieval, precision is the fraction of retrieved documents that are relevant to the search and is computed as:

$$\frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

Recall: Recall in information retrieval is the fraction of the documents that are relevant to the query that are successfully retrieved and is computed as:

$$\frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|}$$

Resource: an object or entity that is accessed or modified within the operation of the software, such as memory, CPU, files, or sockets. Resources can be system-level (memory or CPU), code-level (function or variable), or application-level (cookie or message). [42]

Semantic Template: the generalized patterns of relationship between software elements and faults, and their association with known higher level phenomena in the security domain. This is not related to various uses of the term "semantic template” in other literature, such as semantic web, website design and natural language processing.

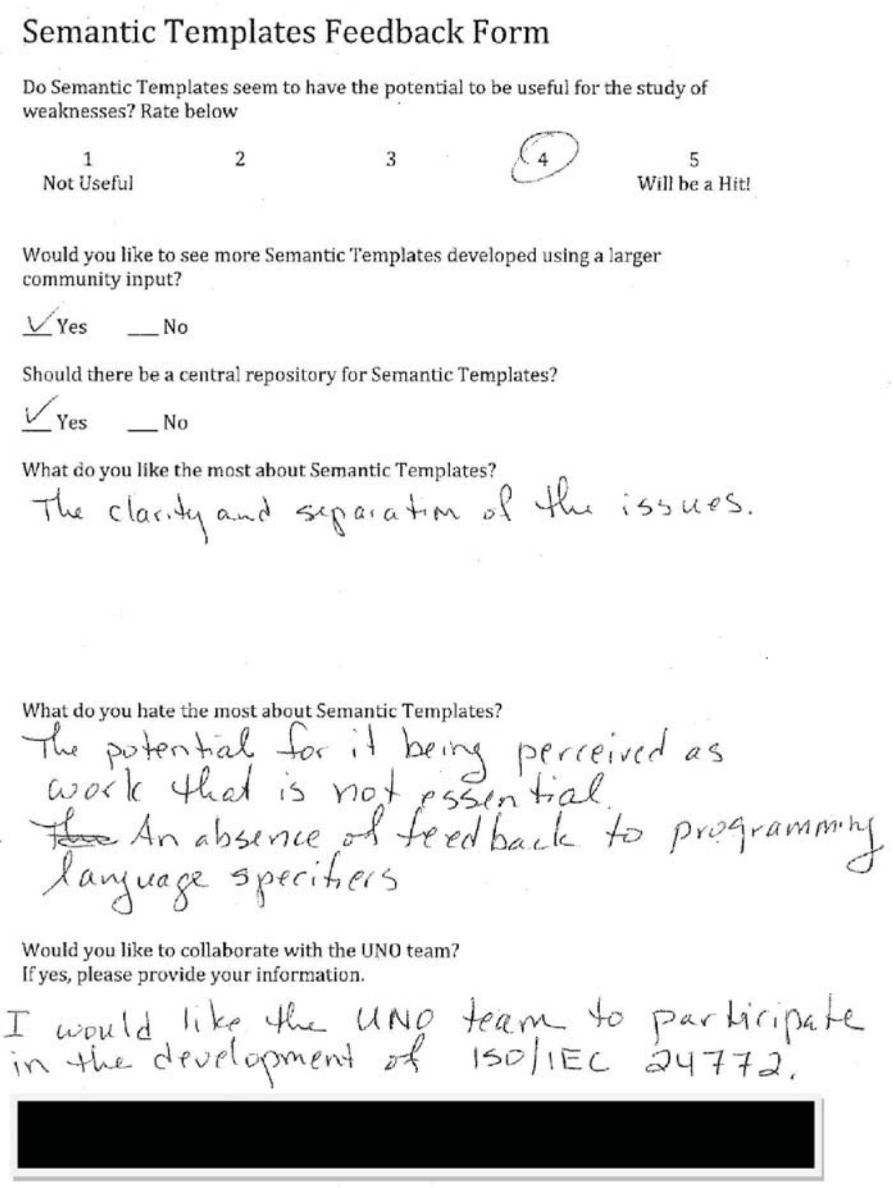
Software Fault: An incorrect step, process, or data definition in a computer program. [IEEE Standard Glossary of Software Engineering Terminology].

TF-IDF: The tf-idf weight (term frequency–inverse document frequency) is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. [Wikipedia]

Vulnerability: an occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness. [42]

Weakness: a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC. [42]

B. Expert Survey Results



Semantic Templates Feedback Form

Do Semantic Templates seem to have the potential to be useful for the study of weaknesses? Rate below

1	2	3	4	5
Not Useful			[✓ 4]	Will be a Hit!

Would you like to see more Semantic Templates developed using a larger community input?

[✓ Yes] ___ No

Should there be a central repository for Semantic Templates?

[✓ Yes] ___ No

What do you like the most about Semantic Templates?

The clarity and separation of the issues.

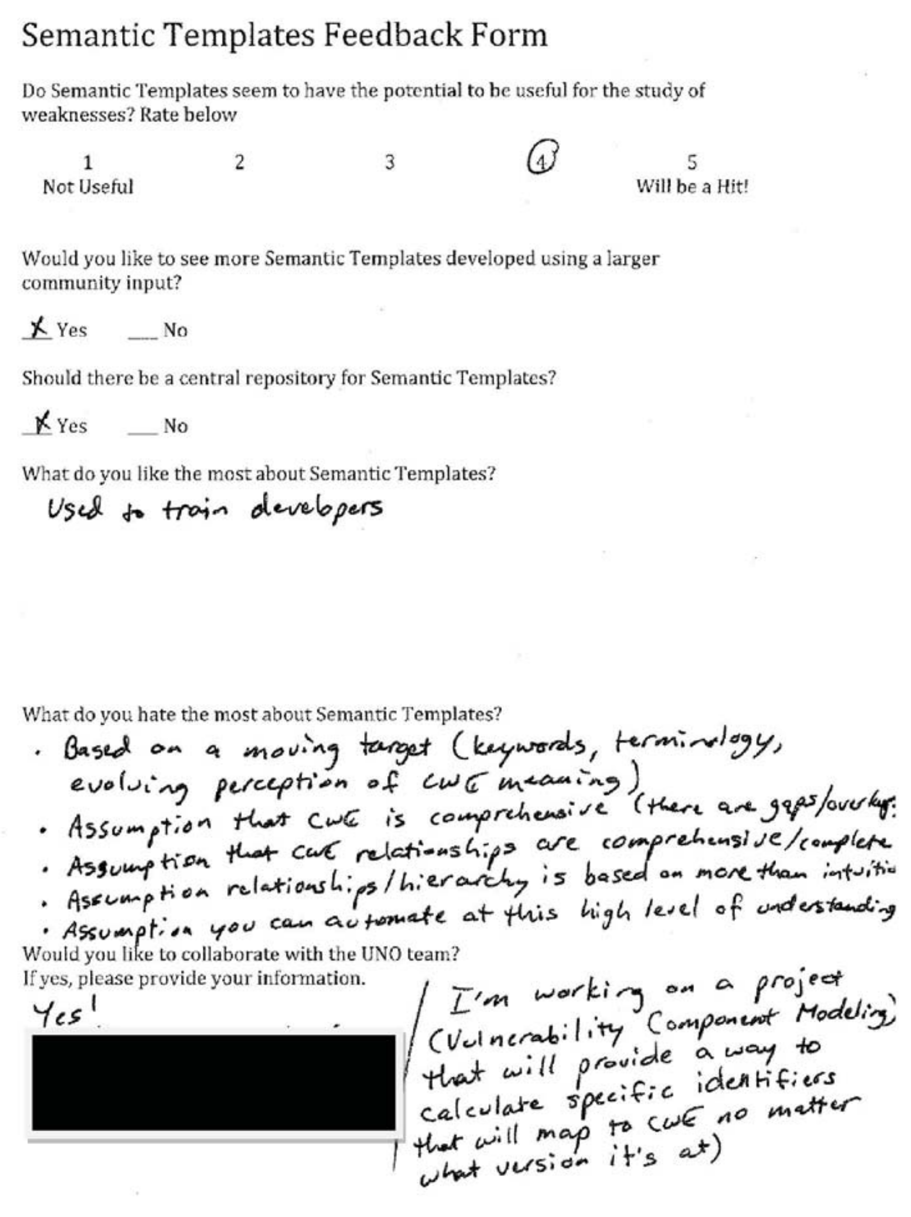
What do you hate the most about Semantic Templates?

- The potential for it being perceived as work that is not essential.
- ~~The~~ An absence of feedback to programming language specifiers

Would you like to collaborate with the UNO team?
If yes, please provide your information.

I would like the UNO team to participate in the development of ISO/IEC 24772.

[redacted]



Semantic Templates Feedback Form

Do Semantic Templates seem to have the potential to be useful for the study of weaknesses? Rate below

1 Not Useful	2	3	[✓ 4]	5 Will be a Hit!

Would you like to see more Semantic Templates developed using a larger community input?

[✓ Yes] ___ No

Should there be a central repository for Semantic Templates?

[✓ Yes] ___ No

What do you like the most about Semantic Templates?

Used to train developers

What do you hate the most about Semantic Templates?

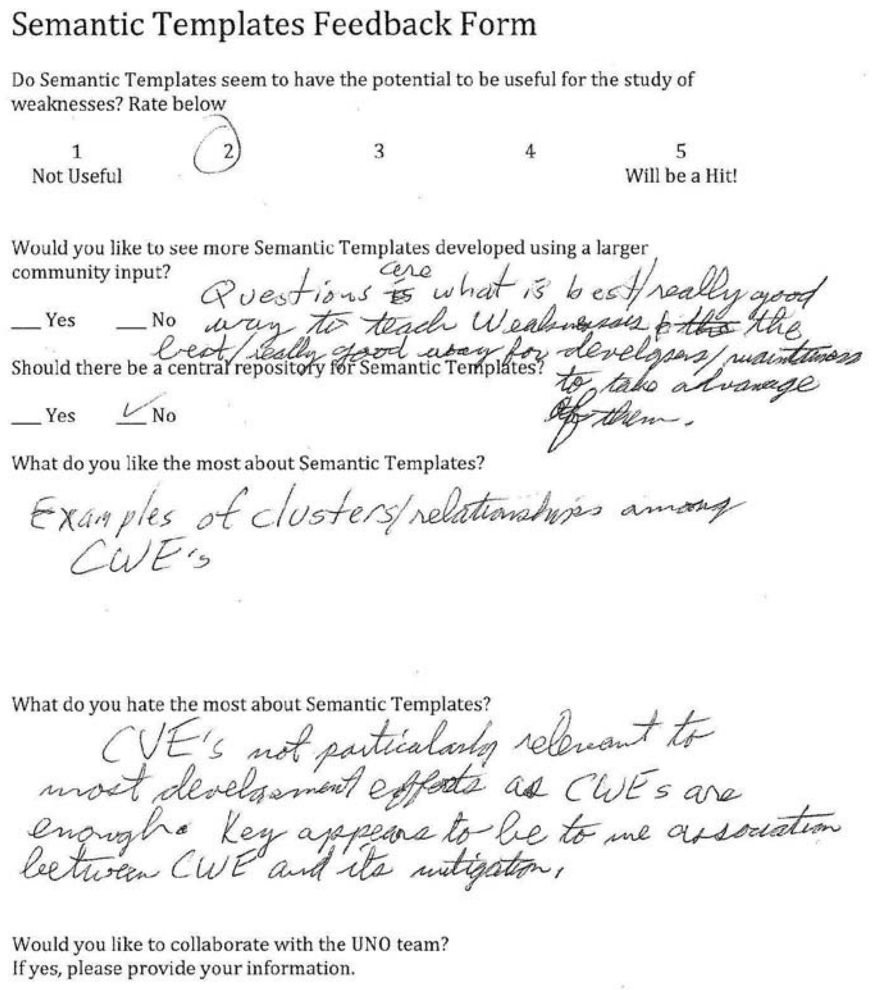
- Based on a moving target (keywords, terminology, evolving perception of CWE meaning)
- Assumption that CWE is comprehensive (there are gaps/overlap:
- Assumption that CWE relationships are comprehensive/complete
- Assumption relationships/hierarchy is based on more than intuition
- Assumption you can automate at this high level of understanding

Would you like to collaborate with the UNO team?
If yes, please provide your information.

Yes!

[redacted]

I'm working on a project (Vulnerability Component Modeling) that will provide a way to calculate specific identifiers that will map to CWE no matter what version it's at)



Semantic Templates Feedback Form

Do Semantic Templates seem to have the potential to be useful for the study of weaknesses? Rate below

1 Not Useful [✓ 2] 3 4 5 Will be a Hit!

Would you like to see more Semantic Templates developed using a larger community input?

__Yes __No

Questions ~~is~~ are what is best/really good way to teach Weaknesses & ~~the~~ the best/really good way for developers/maintainers to take advantage ~~of~~ of them.

Should there be a central repository for Semantic Templates?

__Yes [✓ No]

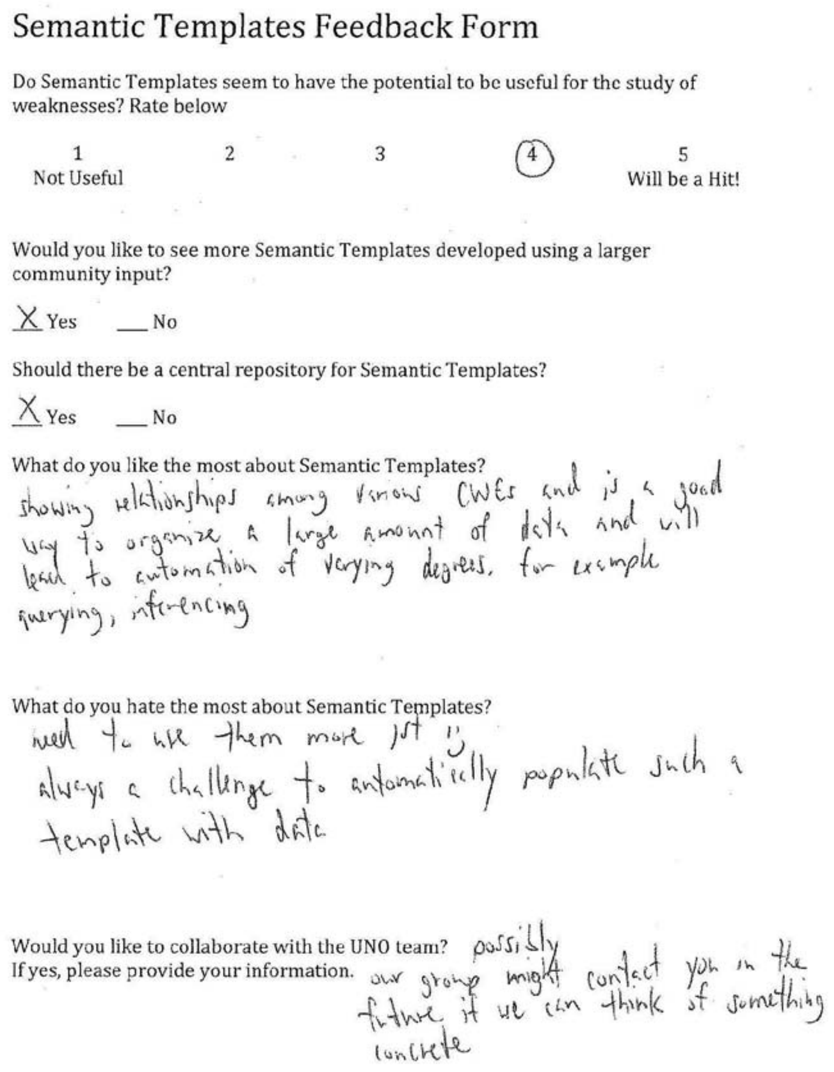
What do you like the most about Semantic Templates?

Examples of clusters/relationships among CWE's

What do you hate the most about Semantic Templates?

CVE's not particularly relevant to most development efforts as CWEs are enough. Key appears to be to me association between CWE and its mitigation,

Would you like to collaborate with the UNO team?
If yes, please provide your information.



Semantic Templates Feedback Form

Do Semantic Templates seem to have the potential to be useful for the study of weaknesses? Rate below

1	2	3	4	5
1 Not Useful	2	3	[✓ 4]	5 Will be a Hit!

Would you like to see more Semantic Templates developed using a larger community input?

[✓ Yes] ___ No

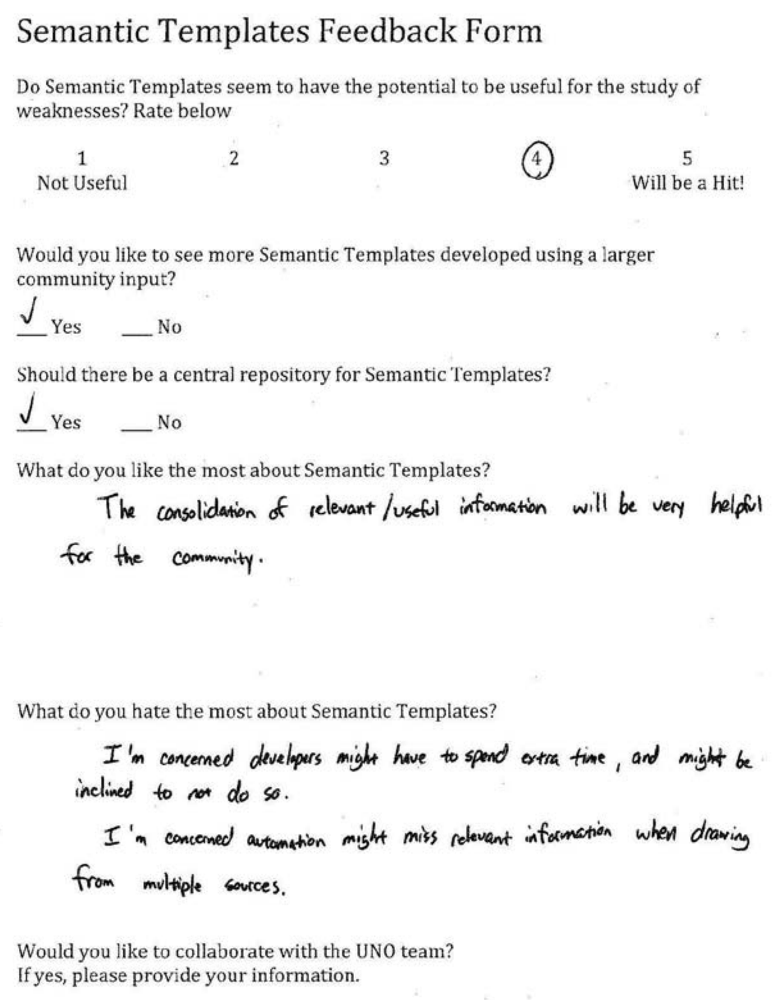
Should there be a central repository for Semantic Templates?

[✓ Yes] ___ No

What do you like the most about Semantic Templates?
showing relationships among various CWEs and is a good way to organize a large amount of data and will lead to automation of varying degrees. for example querying, inferencing

What do you hate the most about Semantic Templates?
need to use them more jst ;)
always a challenge to automatically populate such a template with data

Would you like to collaborate with the UNO team? possibly
If yes, please provide your information. our group might contact you in the future if we can think of something concrete



Semantic Templates Feedback Form

Do Semantic Templates seem to have the potential to be useful for the study of weaknesses? Rate below

1 Not Useful	2	3	[✓ 4]	5 Will be a Hit!

Would you like to see more Semantic Templates developed using a larger community input?

[✓ Yes] ___ No

Should there be a central repository for Semantic Templates?

[✓ Yes] ___ No

What do you like the most about Semantic Templates?

The consolidation of relevant/useful information will be very helpful for the community.

What do you hate the most about Semantic Templates?

I'm concerned developers might have to spend extra time, and might be inclined to not do so.

I'm concerned automation might miss relevant information when drawing from multiple sources.

Would you like to collaborate with the UNO team?
If yes, please provide your information.