8-12-2016

A Method for Recommending Computer-Security Training for Software Developers

Muhammad Nadeem

A method for recommending computer–security training for software developers

By

Muhammad Nadeem

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

August 2016

A method for recommending computer–security training for software developers

By

Muhammad Nadeem

Approved:

Byron J. Williams
(Major Professor)

David A. Dampier
(Committee Member)

Gary Bradshaw
(Committee Member)

Robert Wesley McGrew
(Committee Member)

T.J. Jankun-Kelly
(Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Muhammad Nadeem

Date of Degree: August 12, 2016

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Byron J. Williams

Title of Study: A method for recommending computer–security training for software developers

Pages of Study: 99

Candidate for Degree of Doctor of Philosophy

Vulnerable code may cause security breaches in software systems resulting in financial and reputation losses for the organizations in addition to loss of their customers' confidential data. Delivering proper software security training to software developers is key to prevent such breaches. Conventional training methods do not take the code written by the developers over time into account, which makes these training sessions less effective. We propose a method for recommending computer–security training to help identify focused and narrow areas in which developers need training. The proposed method leverages the power of static analysis techniques, by using the flagged vulnerabilities in the source code as basis, to suggest the most appropriate training topics to different software developers. Moreover, it utilizes public vulnerability repositories as its knowledgebase to suggest community accepted solutions to different security problems. Such mitigation strategies are platform independent, giving further strength to the utility of the system.

This research discussed the proposed architecture of the recommender system, case studies to validate the system architecture, tailored algorithms to improve the performance of the system, and human subject evaluation conducted to determine the usefulness of the system.

Our evaluation suggests that the proposed system successfully retrieves relevant training articles from the public vulnerability repository. The human subjects found these articles to be suitable for training. The human subjects also found the proposed recommender system as effective as a commercial tool.

Key words: Recommender system, software security, software vulnerabilities, CWE, NVD, training, tf–idf, Jaccard index, Static code analysis, FindBugs.

DEDICATION

I dedicate this work to my adorable kids, Simra Nadeem, Ifra Nadeem, and Tabish Nadeem, who spent six precious years of their childhood away from me.

ACKNOWLEDGEMENTS

I feel obliged to express my gratitude to several people who have contributed to completion of this research. First and foremost, I would like to thank to Dr. Byron J. Williams, my major professor and dissertation director, for his continuous support, supervision, mentoring, and guidance throughout the course of this research. I would also like to sincerely thank Dr. Edward B. Allen, my former advisor, for his continuous guidance and diligent technical feedback to move this research in the right direction since the beginning.

I would also like to thank my dissertation committee members, Gary L. Bradshaw, David A. Dampier, and R. Wesley McGrew for their valuable guidance in conducting human subject experiments, statistical analysis of data, and finalizing the architecture of the recommender system.

I would like to thank the Dr. Mohamed El–Attar, Zadia Codabux, Kazi Zakia Sultana, Tanmay Bhomik, James Thompson, Ibrahim Abdoulahi, and other members of Empirical Software Engineering Group for their valuable feedback on my research. I would also like to thank all those who volunteered for participation in the human subject evaluation.

I would also like to thank Dr. Donna S. Reese, head of Computer Science and Engineering department, who not only provided me the opportunity to attend Preparing Future Faculty program and other academic workshops but also has been a source of inspiration and motivation.

I am highly grateful to the United States Department of State, the Fulbright Program, the International Institute of Education (IIE), and the Higher Education Commission of Pakistan for the financial support. Also, the Department of Computer Science and Engineering, for the partial financial support through Bridge Fellowship program.

Last but not the least, I would like to thank all my friends and family members who encouraged, supported, and motivated me throughout the course of my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Research Hypothesis

The following is the hypothesis for our research.

> Potential vulnerabilities in an individual developer's source code can be automatically mapped to relevant articles in public vulnerability repositories that are suitable for training the developer regarding vulnerability mitigation.

1.2 Understanding the Problem

Security breaches in software systems are often caused by vulnerable code, which result in loss of confidential data in addition to reputation and financial damages. A report[1] published by Mcafee Inc. reveals that the world wide annual losses due to security breaches are $\$375 - \$575$ billion. Moreover, a report[2] by Microsoft suggests that a defect found and fixed, especially later in the development or post–development process, may cost 100 times or even higher.

To achieve robust software security, developers must be given proper training on secure coding practices. Conventional training methods are limited as they do not take the prior code written by the developers into account.

[1] See `http://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime2.pdf` (accessed Jan. 20, 2015).

[2] See `http://www.microsoft.com/hun/getthefacts/ESGNov_2006SDL.mspx` (accessed Jan. 20, 2015).

The cost of fixing security vulnerabilities in later stages of software development life-cycle can be exponentially high. Helping software developers to write secure code in the first place is a key to avoid such costs.

1.3 Motivation and Objective

The motivation for this research is the idea that training for each software developer should be in individually identified and focused areas of software security rather than a generalized security training for all developers. The main objective of our proposed system is to recommend precise and focused training articles to software developers. Our approach intends to help developers avoid unsafe coding practices in an efficient and effective way.

1.4 Our Approach to Solving the Problem

Our approach relies on analyzing the code written or modified by individual developers using static analysis tools[3] e.g., FindBugs,[4] which detect security and other vulnerabilities present in the source code. We use the flagged vulnerabilities as the basis for identifying most relevant training articles for developers who contributed to writing those chunks of code, hence improving their skill in terms of software security. We explain our approach with the help of a simple example in Figure 1.1.

It is important to mention that accurately identifying vulnerabilities in developers code is the key to the success of proposed system.

[3]List of tools for static code analysis, `http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis` (accessed Sep. 26, 2014).

[4]FindBugs, `http://findbugs.sourceforge.net/` (accessed Sep. 26, 2014).

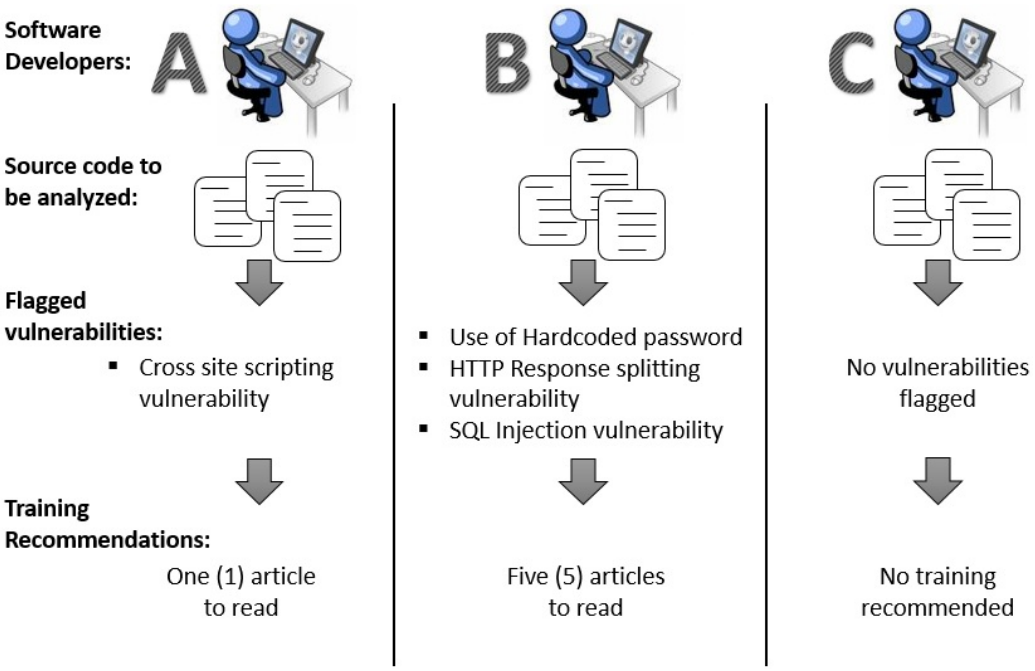


Figure 1.1

Working of proposed system

1.5 Defining the Recommender Systems

Recommender systems are a special type of information filtering system, which seek to predict the rating or preference that a user would give to an item [50]. Recommender systems have been widely used in recent years for various purposes. Common applications of recommender systems include recommending music [29] or movies [5] based on users' demographics and interests and recommending books, magazines [47], or news articles [22] to potential readers or subscribers. Recommenders systems are also popular in social media where they recommend personalized social updates [45] and social tags [7]. Online merchants like Amazon boost sales by recommending appropriate products [63] and services to their potential customers. In addition to the categories described above, there are systems for recommending restaurants, insurance, and financial services.

The effectiveness of recommender systems across such a broad range of applications suggested a novel application: Providing focused and precise security training to software developers based on the past coding practices of each developer. Such training can be more effective and efficient than the conventional generalized security training in terms of helping developers to avoid unsafe coding practices.

1.6 Leveraging the Power of Open Source

The proposed recommender system makes use of valuable knowledge items in vulnerability repositories, such as Common Weakness Enumeration, CWE, which are contributed by software security experts across the globe, and are available for public use for free. CWE repository is a unified, measurable set of software weaknesses. It enables more ef-

fective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code.

With the help of a script, the proof-of-concept implementation of the proposed recommender system downloaded and stored more than 700 articles from CWE repository. Each article details a specific software weakness such as Cross Site Scripting, SQL injection, use of hard–coded passwords, and buffer overflow.

The structure of each CWE article is same, and contains information on potential mitigation for the vulnerability along with other useful information, more details on the structure of CWE article can be found in later sections. The potential mitigation strategies in CWE articles may serve as a training resource for the developers. The proof–of–concept recommender system maps the discovered vulnerabilities in the source code to the CWE articles to identify relevant articles which may be used to train different software developers.

Static code analysis [64], which is an emerging technique for secure software development, analyzes large software code bases, without execution, to reveal potential vulnerabilities present in the code. Static analysis can evaluate the system or a component of a system based on its form, structure, content, or associated documentation. It is analogous to the analysis of development products that rely on human examination. The proposed recommender system utilizes open source static analysis tool(s) as one of it's components.

Though the proof–of–concept recommender system uses only the CWE repository, the proposed research will also consider other vulnerability repositories.

1.7 Organization

This document is organized int the following way. Chapter 2 discusses related work, chapter 3 discusses the open source, commercial, and custom built tools used in this research, chapter 4 outlines the research methodology, chapter 5 discusses the architecture of the proposed recommender system, chapter 6 presents the preliminary case study, chapter 7 discusses the human subject evaluation of the recommender system, chapter 8 details the tailored approach and compares them with the original approaches discussed in previous chapters, chapter 9 is based on discussion, and chapter 10 presents the conclusion and future work.

CHAPTER 2

RELATED WORK

In this chapter we discuss the systematic mapping study on recommender systems for software engineering, we also discuss potentially useful information retrieval algorithms, and provide background information on different security vulnerabilities.

2.1 Recommender Systems for Software Engineering

The idea of recommender systems is not new for the software engineering domain. Several recommender systems have been proposed or built to facilitate software developers to perform various tasks. We briefly describe some well known recommender systems below.

2.1.1 Debug Adviser

Debug Adviser [3] is a recommender system, which helps developers to fix software bugs. A developer enters the contextual information of the bug into the tool, and the tool searches through the repository of projects for bugs with similar context which have already been fixed. It requries the developer to manually enter the bug information while we make use of static analysis techniqes instead. Moreover, the scope of Debug Adviser

is limited to the project repository, while our proposed system makes use of vulnerability repositories to recommend potential mitigation strategies.

2.1.2 Strathcona

Strathcona [26] is a recommendation tool to assist developers in finding relevant fragments of code of an API's use. The tool recommends examples that can be used by developers to provide insight on how they are supposed to interact with the API. Like our system, the purpose of the Strathcona is to train developer, but its scope is focusing on use of specific APIs.

2.1.3 Hipikat

Hipikat [17] is a recommender tool that scans a project's archives and recommends artifacts from the archive that are related to the software developer's task at hand. The artifacts can be versions of source code modules, bugs reports, archived electronic communications, and web documents. Hipikat is especially helpful for open source projects.

2.1.4 RASCAL

RASCAL [37] is a recommender agent that tracks usage histories of a group of developers to recommend to an individual developer components that are expected to be needed by that developer based on the experiences of the groups. RASCAL works in the same way as Hipikat to help new individuals to adjust in group setting efficiently, but it is not specifically related to security training.

2.1.5 Recommender System for Requirements Elicitation

Requirements elicitation includes complex tasks for which recommender systems can be helpful, examples of such tasks are identification of potential subject matter experts, Recommender System to Support Requirements Elicitation [13] recommends possible features for stakeholders to consider, and keeping stakeholders informed of relevant issues.

2.1.6 Contextual Recommendations

Context Based Recommendations for Software Developers [2] help software developers to use the elements of the source code in the integrated development environment (IDE) in smarter ways by identifying the elements that are commonly related. The system provide an efficient way to reach the desired elements. This system is significantly different from our proposed system as it helps developer only to identify related elements of source code and use them effectively.

2.2 Systematic Mapping Study

To get a detailed insight of the existing recommender systems related to the domain of software engineering, we conducted a systematic mapping study. This sections details our mapping study.

The following creteria were used to filter the articles.

- Articles that do not address core software engineering functions were excluded.
- Mapping studies on recommender systems were excluded.
- User studies on recommender systems were excluded.
- Duplicate studies were excluded.

Table 2.1

Filtering Creteria

Filter criteria	Excluded	Remaining
Total articles found using search criteria		927
Filtered to remove duplicate studies	141	786
Filtered based on title of the article	653	133
Filtered after reading abstract of the article	72	61
Filtered after reading conclusions and results	7	54

The summary of filtering process is given in Table 2.1.

After conducting the systematic mapping study, we categorized the recommender systems based on their function, they are summarized in Table 2.2.

In addition to the categories of software engineering recommender systems shown in Table 2.2, there are other systems for miscellaneous tasks e.g., recommender system for identifying critical modules for rigorous testing [30], automated test recommendation [28], selection of software development life cycle [32], assigning tasks (e.g., bug reports) to software developers [36], envisioning a holistic view of recommender systems for software engineering [48].

The recommender systems discussed in this section facilitate individual software developers in performing various software engineering tasks. However, we did not find literature on recommender systems which, by utilizing static code analysis or otherwise, recommend security related training topics to software developers.

More details about systematic mapping study can be found in the Appendix A.

Table 2.2

Recommender Systems for Software Engineering

Recommender system category	Related systems
Information retrieval	
They facilitate software developers identifing relevant information and artifacts based on the context e.g., source code snippets, software components for reuse, and API usage examples.	[2], [6], [16], [17], [18], [20], [21], [23], [24], [25], [33], [35], [37], [39], [52], [54], [60]
Software development	
They enhance development environment for software developers e.g., identifying tools or commands to use in an IDE, recommending meaningful method names.	[1], [31], [40], [61]
Software project management	
They facilitate software project management activities e.g., effort estimation, identifying project team, assigning mentors to software project newcomers.	[15], [46], [55], [56]
Requirements specification	
They help in requirements specification, e.g., identifying non-functional requirements based on context.	[10], [14], [19], [53], [62]
Software design and architecture	
Recommender systems that facilitate maintaining software architecture, e.g., identifying architectural violations in software, recommending design patterns.	[11], [44], [51], [58]
Other/ Miscellaneous	
	[28], [30], [32], [36], [48]

2.3 Mapping Algorithms

Mapping algorithms are used to find similarity between two given documents, in this research we intend to calculate similarity between the description of vulnerabilities and the articles in the public vulnerability repositories. We intend to use the following mapping algorithms.

1. Vector Space Model (VSM) with term frequency–inverse document frequency (tf–idf) [9].
2. Jaccard index or Jaccard similarity coefficient [57].

The Jaccard coefficient measures similarity between finite sample sets A and B.

$$J(A, B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid} \tag{2.1}$$

As it is obvious from equation (2.1) that the Jaccard coefficient does not take the weight of the terms into account. However the tf–idf scores are offset by the frequency of the words in the corpus, which helps to control the fact that some words are generally more common than others. More details on tf–idf can be found in chapter 5.

2.4 Software Vulnerabilities

A vulnerability is a weakness in a software system that may be exploited by an attacker. Some examples of vulnerabilities are Injection Attacks, Broken Authentication and Session Management, Cross-Site Scripting (XSS), Insecure Direct Object References, Security Misconfiguration, Sensitive Data Exposure, Missing Function Level Access Control, Cross-Site Request Forgery (CSRF), Using Components with Known Vulnerabilities, and

Unvalidated Redirects and Forwards. These examples have been taken from OWASP[1] Top Ten Project for the year 2013.

Exploiting these vulnerabilities results in financial and reputation losses to the organizations in addition to the voilation of privacy of the individuals effected. Injection attacks alone have caused several major data breaches to financial and healthcare organizations in recent years. There are several preventive and reactive approaches to counter such vulnerabilities, the work proposed in this research falls under the first category.

The code repository analyzed in our experiments involves four security vulnerabilities. They are discussed briefly below.

2.4.1 Cross Site Scripting (XSS)

The Cross Site Scripting, or XSS, vulnerability allow an attacker to add malicious code to a web application. When the infected application is accessed by victim, a legitimate user, the malicious code/script executes on user's web browser. The working of Cross Site Scripting[2] is explained in Figure 2.1.

The malicious code runs on victim's computer everytime the infected webpage is accessed. Common consequences of XSS attack are session hijacking and stealing victim's confidential information.

[1]OWASP, `https://www.owasp.org/` (accessed Nov 21, 2014).

[2]Typical XSS attack, `http://www.pitsolutions.ch/blog/cross-site-scripting/` (accessed Jun 08, 2016).

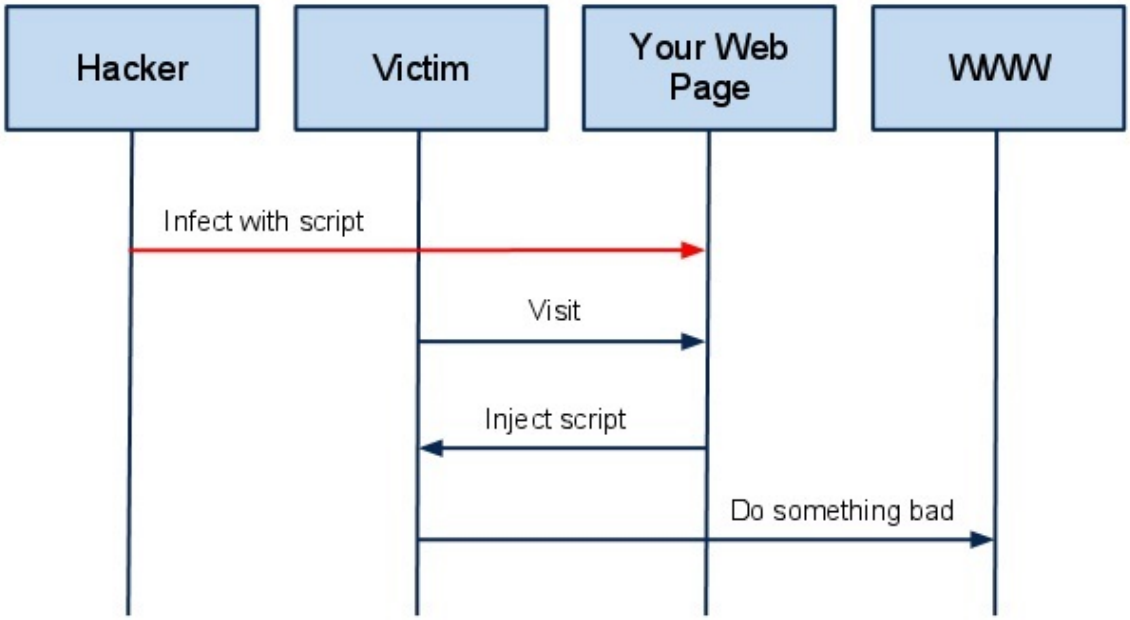


Figure 2.1

Typical Cross Site Scripting (XSS) Attack

2.4.2 Hardcoded Constant Database Password

Source code responsible for establishing connection to the database engine may have hardcoded database password, anyone with access to either the source code or the compiled code can easily learn the password, which makes the entire application/database vulnerable. Following is an example[3] of such vulnerable code.

```
DriverManager.getConnection(url, "scott", "tiger");
```

In some cases, if establishing connection to the database fails, the application may accidently display the hardcoded username and password to users through web browser.

[3]CWE - Hardcoded passwords, `https://cwe.mitre.org/data/definitions/259.html` (accessed Jun 08, 2016).

However, this is only possible if the application configuration permits its users to read detailed error messages.

2.4.3 HTTP Response Splitting

The HTTP response splitting[4] occurs when data enters a web application through an untrusted source (the attacker). The web server does not neutralize or incorrectly neutralizes `CR` and `LF` characters before the data is included in outgoing HTTP headers.

Including unvalidated data in an HTTP header allows an attacker to specify the entirety of the HTTP response rendered by the browser. When an HTTP request contains unexpected `CR` (carriage return, also given by `%0d` or `\r`) and `LF` (line feed, also given by `%0a` or `\n`) characters the server may respond with an output stream that is interpreted as two different HTTP responses (instead of one). An attacker can control the second response and mount attacks such as cross-site scripting and cache poisoning attacks.

2.4.4 SQL Injection

A program becomes vulnernable to SQL injection attack if it allows a nonconstant string to be passed to execute method of an SQL statement.

SQL injection attacks (SQLIAs) are launched by entering malicious characters into the input fields of web applications resulting in a modified SQL query [42]. The concept of SQL injection attack is explained in Figure 2.2.

[4]CWE - HTTP response splitting, `https://cwe.mitre.org/data/definitions/113.html` (accessed Jun 08, 2016).

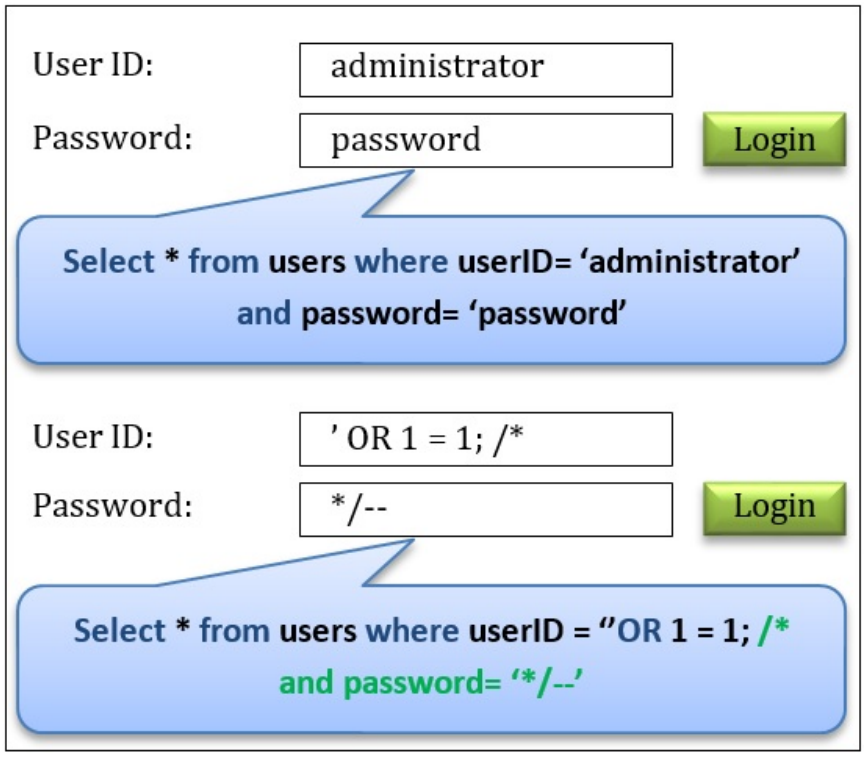


Figure 2.2

Example of SQL injection attack

SQL injections have emerged as one of the most dangerous types of attacks to web-based systems and are ranked number one among the Open Web Application Security Project's (OWASP) top ten vulnerabilities.

CHAPTER 3

TOOLS

This research involves use of different commercial, open source, and custom built tools and APIs. In this chapter we briefly discuss these tools and APIs.

3.1 Tools for Static Analysis

There are plenty of proprietary and open source tools available which can analyze millions of lines of code and uncover various types of vulnerabilities. These vulnerabilities include but are not limited to SQL injections, buffer overflows, cross site scripting, improper security settings and information leakage [27].

Some tools can analyze code written in multiple languages while others are language specific. Similarly, tools may differ with respect to the purpose they are built, some tools scan the code for coding style, for example, proper comments and naming conventions, while other look for vulnerabilities only.

To analyze the selected source code repositories, we plan to use the following tools.

- `FindBugs` analyzes Java codebases and finds several different types of potential vulnerabilities in the code. The tool was created at the University of Maryland in 2003. The tool is freely available for noncommercial use.
- `HP Fortify Static Code Analyzer` helps verify that software is trustworthy. It also helps developers identify software security vulnerabilities in a variety of programming languages which include C, C++, Java, JSP, ASP/ASP.NET, PHP, Visual Basic, VBScript, JavaScript, PL/SQL, COBOL. In addition it has the capability

to scan configuration files. To verify that the most serious issues are addressed first, it correlates and prioritizes results to deliver an accurate, risk-ranked list of issues.

Although these tools provide an efficient way of scanning large code repositories for vulnerabilities, usually a high percentage of discovered vulnerabilities are actually false positives [4] namely, discovered vulnerabilities are not actually present in the code. Software development teams can spend many man-hours to track and fix the discovered vulnerabilities. The results generated by these tools can cause unproductive consumption of time and resources [8].

3.2 Tools for Mapping Algorithms

`SimScore` is a .NET platform based implementation to calculate two statistics namely tf–idf [9] and Jaccard index or Jaccard similarity coefficient [57]. This tool will be used to calculate the similarity scores between the description of flagged vulnerabilities and the articles in public vulnerability repositories. SimScore has been developed by the Empirical Software Engineering Research Group of the Computer Science and Engineering Department at Mississippi State University [34].

3.3 Part of Speech (POS) Tagger

We use the Part of Speech (POS) Tagger[1],developed by Stanford University, reads text in a given document and assigns parts of speech to each word.

[1]POS Tagger, `http://nlp.stanford.edu/software/tagger.shtml` (accessed Apr 20, 2016).

3.4 LOC Calculator

We use `LOC Calculator 1.2` which is a simple Java based tool for calculating lines of code of source files. It scans a single file or a directory, it has option to ignore white spaces. It ignores binary files.

3.5 Custom Built Tools

We use custom built tools for different purposes. All custom built tools have been made publically available to facilitate other researchers. We briefly discuss the custom built tools below.

3.5.1 Article Extractor

The `Article Extractor` extracts all the articles from the dictionary view of the CWE[2] repository and stores then in individual files. These articles serve as knowledgebase for our recommender system.

3.5.2 Noun Extractor

This custom built tool has two important functions. First, it uses the Part of Speech (POS) Tagger library to assign a part of speech to each word, such as noun, adjective, verb etc., and creates a tagged document for every input document. Second, it reads the tagged documents one by one, extracts nouns from the doucment, and stores them in a new file.

[2]CWE, `http://cwe.mitre.org/` (accessed Apr 30, 2014).

3.6 Public Vulnerability Repositories

A public vulnerability repository contains information on vulnerabilities, potential mitigations, demonstrative examples, consequences, etc. These repositories are maintained and made available for public use. They provide unified, effective, and standard information on security vulnerabilities. Examples of such repositories are National Vulnerability Database, NVD,[3] Common Vulnerabilities and Exposures, CVE,[4] and Common Weakness Enumeration, CWE.These repositories are frequently updated and some of them do provide publically accepted solution along with other useful information on various software security problems. The proposed recommender system leverages the power of public vulnerability repositories by mapping the flagged vulnerabilities in source code to the articles in these repositories.

[3]NVD, `https://nvd.nist.gov/` (accessed Apr 25, 2014).
[4]CVE, `http://cve.mitre.org/` (accessed Apr 30, 2014).

CHAPTER 4

RESEARCH METHODOLOGY

4.1 Hypothesis and Research Questions

Following is the hypothesis for our research:

> Potential vulnerabilities in an individual developer's source code can be automatically mapped to relevant articles in public vulnerability repositories that are suitable for training the developer regarding vulnerability mitigation.

Conventional training techniques do not take the code written by the developers over time into account, which makes training less effective. Static analysis helps identify the security vulnerabilities in source code written by developers. Using appropriate mapping algorithms, the flagged vulnerabilities can be mapped to the articles in public vulnerability repositories. These articles contain useful information which may be used to train developers to write secure and robust code.

To generalize the core idea explained above, we conduct case studies on a large scale open source system. There is evidence that the selected open source system has been used by the research community for similar studies.

4.2 Research Questions

This dissertation attemps to answer the following research questions:

1. How do the recommendations based on CWE repository compare to recommendation from HP Fortify in terms of training suitability?

2. How do the TF-IDF and Jaccard index algorithms compare to each other to identify relevant articles in public repositories given the evaluation of human subjects?
3. Regarding tailored approaches,
    (a) How does the tailored mapping algorithm based on TF-IDF scores and additional attributes, i.e., recency and completeness of the articles compare, to the other algorithms in terms of relevance?
    (b) How does the tailored mapping algorithm based on Jaccard index scores and additional attributes, i.e., recency and completeness of the articles, compare to the other algorithms in terms of relevance?
4. What is the statistical profile of developers in the case study based on the following-parameters?
    (a) Tendency to cause vulnerabilities over the case study window
    (b) Recency of induced vulnerabilities
    (c) Severity of induced vulnerabilities

The following sections discuss each research question in detail and lists the experiments conducted to answer each question.

4.2.1 How do the recommendations based on CWE repository compare to recommendation from HP Fortify in terms of training suitability?

Public vulnerability repositories host useful information on software vulnerabilities. This research question addresses whether or not this information can be used for training purposes. We characterize these repositories to find out whether they are suitable as a training resource to help developers write secure and robust code.

We compare the recommended articles from public vulnerability repository, i.e. the CWE repository, with the articles from commercial tool's repository. Detail about experiment design to address this research question can be found in section 4.3.3 and 4.3.4.

4.2.2 How do the TF–IDF and Jaccard index algorithms compare to each other to identify relevant articles in public repositories given the evaluation of human subjects?

Our research primarily uses tf–idf and Jaccard Index based approaches to map flagged vulnerabilities to the articles in public vulnerability repository. To answer this research question we conduct a human subject evaluation. Detail about experiment design to address this research question can be found in section 4.3.4 and 4.3.5.

4.2.3 Research questions regarding tailored approach

The research questions regarding tailored versions of tf–idf and Jaccard Index based approaches are given as under.

4.2.3.1 How does the tailored mapping algorithm based on TF-IDF scores and additional attributes, i.e., recency and completeness of the articles compare, to the other algorithms in terms of relevance?

We create a tailored version of tf–idf based approach and see if we can achieve improvement in terms of finding relevant training articles. The modified approach may include pre–processing the entire dataset in addition to incorporating attributes, i.e., recency and completeness of the articles.

4.2.3.2 How does the tailored mapping algorithm based on Jaccard index and and additional attributes, i.e., recency and completeness of the articles, compare to the other algorithms in terms of relevance?

We create a tailored version of Jaccard index based approach and see if we can achieve improvement in terms of finding relevant training articles. The modified approach may in-

clude pre–processing the entire dataset in addition to incorporating attributes, i.e., recency and completeness of the articles.

Detail about experiment design to address the research questions 4.2.3.1 and 4.2.3.2 can be found in section 4.3.5.

4.2.4 What is the statistical profile of developers in the case study based on the following parameters?

1. Tendency to cause vulnerabilities over the case study window
2. Recency of induced vulnerabilities
3. Severity of induced vulnerabilities

To answer this research question, we analyze different versions of open source software systems for security vulnerabilities and study the trends.

4.3 Experiment Design

This research involves three main experiments. First, the priliminary case study, which is discussed in chapter 6; second, The human subject evaluation of the proposed recommender system discussed in chapter 7; and third, the implementation of a tailored approach discussed in chapter 8. The summary of dataset, pre–processing, and experiment is given below.

4.3.1 Getting Dataset and Tools Ready

For experiments, we need to identify the following.

- Identify an open source systems, to be used as target code, say C.

- Identify and explore a static analysis tools, say T.
- Identify a public vulnerability repository, say V.
- Identify and implement two mapping algorithms, say A_1 and A_2.
- Identify a commercial tool that analyzes a given code repository and recommends training.

The next section discusses how do we pre–process the data.

4.3.2 Pre–processing

Once we identify target source code, tools, and public vulnerability repository, we pre–process data as discussed below:

- Extract information from public vulnerability repository. Store information on each vulnerability in a seperate file, the $training\ articles$.
- Exclude articles without mitigation strategies.
- Extract severity level of each vulnerability (to be used for prioritizing training).

The sections below discuss details about experiments.

4.3.3 Experiment I: Preliminary case study

This proof-of-concept case study validates the architecture of the proposed recommender system with a preliminary empirical evaluation. The study is based on analysis of a open source system using a open source static analysis tool, and uses a tf–idf based approach to map identified vulnerabilities to the articles in public vulnerability repository.

The following experiment will be performed to answer the research question listed above.

- Select three open source large scale real world systems to conduct different experiments explained in the following sections.

- Use two static analysis tools to find out security vulnerabilites in source code of each selected open source system and compare the results for each case study.
- Analyze flagged vulnerabilities for potential training topics for individual developers.

4.3.4 Experiment II: Human Subject Evaluation

The human subject evaluation of the system extends the preliminary study. In this study we use Jaccard index, a set based approach, in addition to tf–idf based approach, for fetching related articles from the public vulnerability repository. Moreover, we also draw a comparison of our proposed system with a commercial tool.

A panel of sofware security experts will assess whether the training material will help the trainee to write secure code in the future? They will be given recommended articles mapped for different vulnerabilities flagged in source code. This will also provide a way to find which public vulnerability repository has better and more well structured information, and also which mapping algorithm has mapped more relevant articles.

The experts are expected to understand the vulnerabilities flagged in the code and the articles in public vulnerability repositories. We intend to involve graduate students and faculty members who fulfill criteria. Students enrolled in Software Security and Software Reverse Engineering classes shall be approached by their respective professors, although they may be required to participate in the study however the may opt to exclude their responce from the data.

The following experiments will be performed to answer the research question listed above.

- Identified experts will analyze and determine whether or not the mitigation strategies recommended by the mapping above for software vulnerabilities are suitable for training?
- Interrater agreement statistics will be calculated.

4.3.5 Experiment III: Implementation of Tailored Algorithms

This study is continuation of the previous studies and implements a tailored version of our approach. We utilize the Part of Speech (POS) Tagger API to extract Nouns from all the documents in dataset (i.e., flagged vulnerabilities and training articles from public vulnerability repository). We also utilize parameters like length of articles in re-ranking the training articles. Lastly, we analyze trends in discovery of vulnerabilities in open source software.

CHAPTER 5

PROPOSED RECOMMENDER SYSTEM ARCHITECTURE

The architecture of the system shown in Figure 5.1 is the context for this research. This is an extension to our preliminary design [38]. The preliminary design relies primarily on the custom built training database whereas the refined architecture leverages the power of vulnerability repositories in addition to custom built training database.

A short description of each module of the proposed system is given below:

5.1 Software Code Repository

The software code repository contains the source code and metadata which is analyzed by the recommender system. Since, the proposed system may use any available static analysis tool(s), which implies that the system is capable of handling source code written in any programming language.

5.2 Static Code Analysis Module

This module contains one or more static analysis tools which scan the given code repository to find vulnerabilities. The output of this phase is an XML based report containing details about detected vulnerabilities (e.g., filename, location, vulnerability descrip-

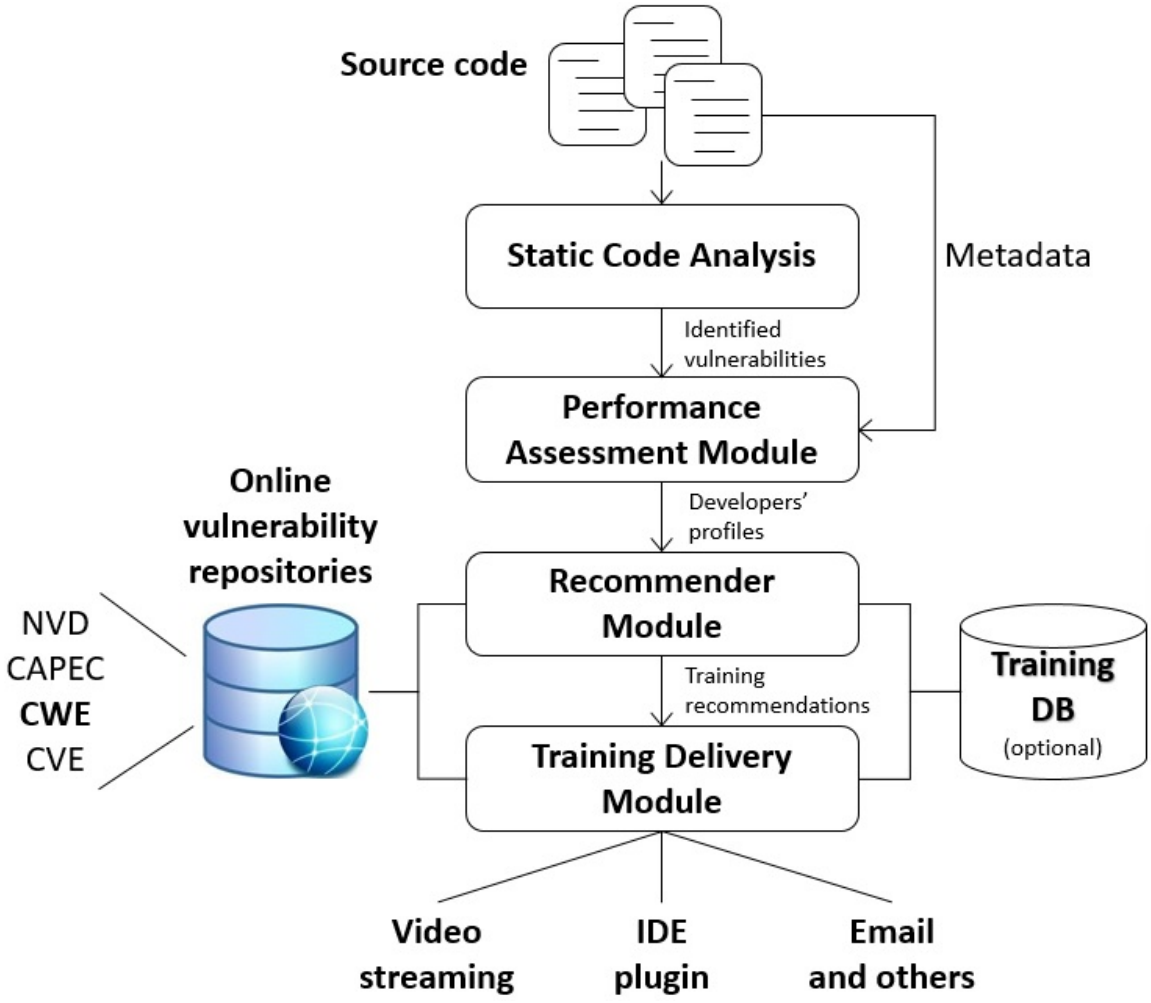


Figure 5.1

Architecture of the Proposed System

tion). Our system is flexible enough to use any tools for static analysis, which makes the proposed system useful for virtually any programming environment.

5.3 Developers Performance Assessment Module

This module takes the results of the static analysis tool(s) and metadata of the software repository as input and creates a profile for each developer. A profile primarily contains the description of vulnerabilities induced by each developer in the code over time. The profiles are fed into the recommendation module.

5.4 Public Vulnerability Repository

This module refers to one or more repositories containing information on vulnerabilities, potential mitigations, demonstrative examples, consequences etc. These repositories are maintained and made available for public use. They provide unified, effective, and standard information on security vulnerabilities.

5.5 Custom Training Database

This refers to a database of custom designed training modules on various topics. This database may also be populated with of-the-shelf training resources. However, this module is optional, as we focus more on using the public vulnerability repositories as our knowledgebase.

5.6 Recommender Module

The Recommender Module takes the profiles of developers as input and calculates the similarity scores between vulnerability descriptions and the articles from vulnerability

repositories. Moreover it queries the custom training database using the information in developers profile to find most appropriate training modules. Output of the module is the recommendation reports customized for individual developers. The Figure 5.2 explains the working recommender module.

5.7 Training Delivery Module

This module accepts the recommendation report from the recommendation module and delivers the training modules to the developers using multiple channels including video streaming, tips delivered via an IDE plugin, and email messages containing helpful literature.

5.8 Intended benefits for practitioners

Practitioners can benefit from proposed recommender system in the following ways:

Getting Started is Easy. The developers don't have to wait for the source code of their current project to become available for analysis; rather, previously written code may be analyzed to obtain precise and sepcific recommendations on secure coding practices.

Preventive Approach to Counter the Security Vulnerabilites. With the help of narrow and precise training recommendations, the practitioners can avoid security vulnerabilites in their source code. This can considerabily lower the effort and cost to fix these vulnerabilities at the later stagesof software development life cycle.

Low Cost Alternative to Commercial Tools. The practitioners can adopt our recommender system which is based on open source tools and public vulnerability repository as a low–cost alternative to commercial tools.

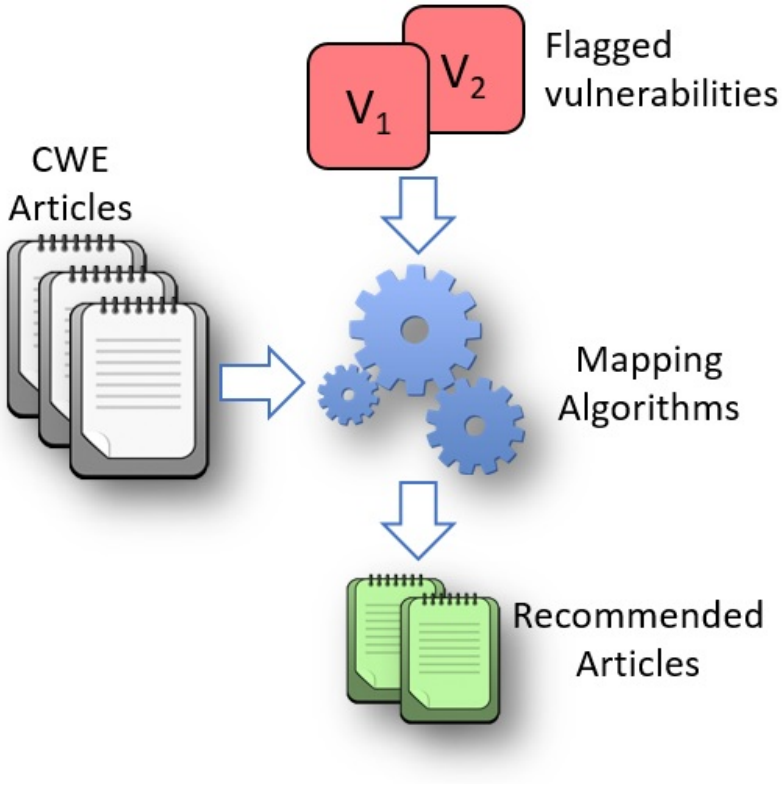


Figure 5.2

Working of Recommender Module

In this chapter we discussed the proposed architecture of computer–security training recommender. It is very important to use the proposed system to analyze large scale open source system. In next chapter, we present a preliminary case study.

CHAPTER 6

PRELIMINARY CASE STUDY

We conducted a proof–of–concept case study [41] to evaluate the feasibility of mapping vulnerabilities discovered by a static analyzer to articles in a public vulnerability repository. These articles generally present mitigation strategies. An open source system called Tolven version 2.0 was the target source code. More information about the system and download instructions may be found on the official website[1]. For static code analysis, we used the open source tool FindBugs[2] version 2.0.3. Code used for analysis is summarized in Table 6.1.

Table 6.1

Summary of System Studied

Item	Description
System analyzed	Tolven 2.0
Number of Java modules	2,957
No. of non-empty lines of code (Calculated using LoC-Calculator 1.2)	417,911

[1]Tolven software downloads, `http://tolven.org/download/` (accessed Feb. 11, 2014)
[2]FindBugs, `http://findbugs.sourceforge.net/` (accessed Sep. 26, 2014).

6.1 Quantitative Analysis

The quantitative study involve analysis of a selected open source system using static code analysis tool "FindBugs". Results of static code analysis are summarized in Table 6.2. Since there can be multiple occurrences of a given vulnerability, the column Flagged Vulnerability types represents distinct vulnerabilities discovered while last column represents Total occurrences for each category.

Table 6.2

Summary of Static Code Analysis

Category	Flagged Defect Types	Total Occurrences
Bad practice	25	195
Correctness	19	64
Dodgy code	20	226
Experimental	2	31
Internationalization	1	105
Malicious code vulnerability	8	323
Multithreaded correctness	1	1
Performance	15	220
Security	4	14
Totals	95	1179

Categories shown in Table 6.2 were generated by the FindBugs 2.0.3 tool. Even though some of these categories seem closely related to each other, the short descriptions of the categories in Table 6.3 show that the categories are distinct.

Every category listed in Table 6.2 has a number of flagged defect types. Due to space limitations, all defect types cannot be listed. However in this study, we analyze the security

Table 6.3

Categories in FindBugs

Category	Description
Bad practice	Code that does not comply with the recommended coding practices.
Correctness	Code that may produce different results than the developer expected.
Dodgy code	Dodgy code means an error prone style of code
Experimental	Code that fails to cleanup steams, database objects, or other objects.
Internationalization	Code that can prevent the use of international character set.
Malicious code vulnerability	Code that can be maliciously altered by other code.
Multithreaded correctness	Code that could cause problems in multithreaded environment.
Performance	Code that could be written differently to improve performance.
Security	Code that can cause possible security problems.

category in detail. The security vulnerabilities along with the number of occurrences are listed in Table 6.4.

Table 6.4

Flagged Security Vulnerability Types

Vulnerability types in Security Category	Occurrences
Cross site scripting vulnerability	11
Hardcoded constant database password	1
HTTP Response splitting vulnerability	1
Non-constant SQL string passed to execute method	1
Total	14

All vulnerability types have a description generated by the FindBugs tool. We extracted a description for each type and stored them in text documents.

We used Vector Space Model (VSM) with term frequency-inverse document frequency (TFIDF) [9] weights to calculate the similarity between the vulnerability descriptions and CWE articles. The indexing process includes removing English stop words from documents. Stemming is then applied to reduce words to their roots, thus improving the effectiveness and the efficiency of the retrieval system [49].

The output of the indexing process is a compact content descriptor, or a profile, where each document is represented as a set of terms:

$$T = \{t_1, t_2, t_3, ..., t_n\}. \tag{6.1}$$

Each term t_i in the set T is assigned a certain weight w_i. Using TFIDF, the weight of each word is calculated as:

$$w_i = tf_i \cdot idf_i \tag{6.2}$$

Where tf_i is the frequency of term t_i in the document and idf_i is the inverse document frequency, and is computed as:

$$idf_i = \log_2(n/df_i) \tag{6.3}$$

Where n is the total number of documents in the collection, and df_i is the number of artifacts in which term t_i occurs. TFIDF determines how relevant a given word is in a particular document. Words that are common in a single or a small group of documents tend to have higher TFIDF, while terms that are common in all documents such as articles and prepositions get lower TFIDF values.

Since the documents are represented as weighted vectors in the N-dimensional space, the similarity of any two documents can be measured as the cosine of the angle between their vectors as follows:

$$Sim(v, a) = \frac{\sum v_i \cdot a_i}{\sqrt{\sum v_i^2 \cdot \sum a_i^2}} \tag{6.4}$$

In equation (6.4), $Sim(v, a)$ is similarity score between vulnerability description v and CWE article a.

The approach discussed above was implemented to automatically calculate similarity scores between descriptions of the flagged vulnerabilities and CWE articles. Both, flagged vulnerability types and CWE articles were stored in text files. There were 95 flagged vulnerability types and 717 repository articles. The mapped articles were filtered based on

the threshold value of 0.35 which was set after carefully reviewing a sample of mapped articles in each category.

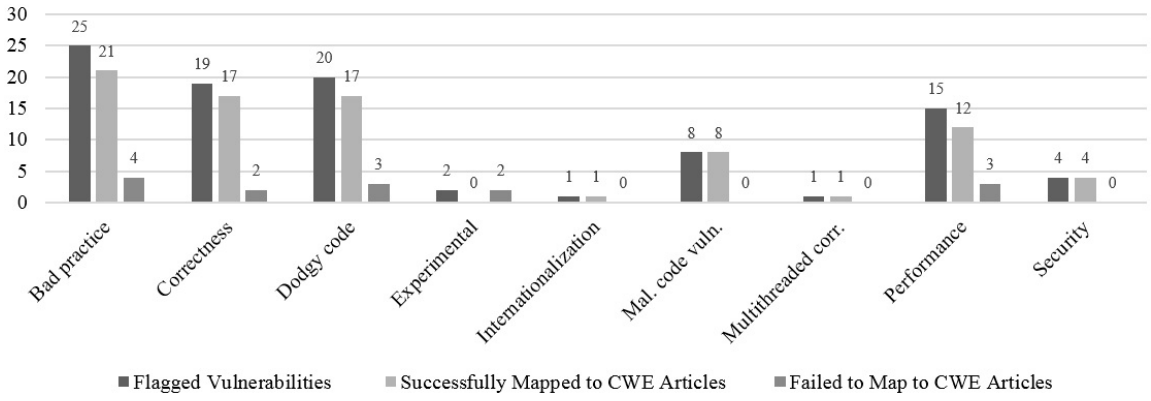


Figure 6.1

Vulnerabilities Mapped to CWE articles

Figure 6.1 shows that 81 out of 95 vulnerability description files were successfully mapped to the related CWE articles. The mapping algorithm failed to find relevant CWE articles for the remaining the 14 vulnerability types. The possible reasons for failing to map these vulnerabilities to CWE articles could be the fact that the status of some of the articles is still Draft or Incomplete. Once the articles in CWE repositories are updated, we expect that the success rate will go higher.

Figure 6.2 shows the mapped CWE articles for each category of vulnerabilities. It can be observed that most of the articles have potential mitigation section. Articles which currently do not, might not be very helpful for training purpose.

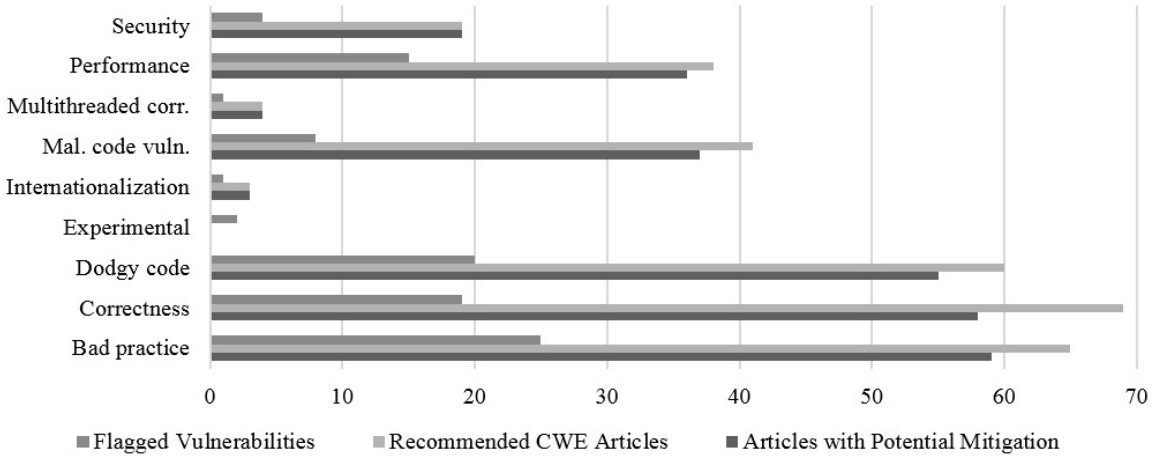


Figure 6.2

Mapped CWE Articles with Potential Mitigations

6.2 Qualitative Analysis

The preliminary qualitative analysis examines only the security category. Table 6.5 provides the details about the recommended CWE articles with their similarity scores and a short description of each recommended article for vulnerability types in the Security category. The mapping algorithm was able to find multiple relevant CWE articles for each flagged vulnerability. The last column provides our qualitative evaluation for relevance of each recommended article. The factors considered while determining the relevance include 'Status of the CWE article' (Draft, Incomplete, and Usable etc.) and 'Opinion of experts'.

It can be observed from the TFIDF scores given in Table 6.5 that CWE articles with higher TFIDF scores have higher relevance to the flagged vulnerability types.

The CWE articles contain highly useful information related to various types of vulnerabilities. This information is usually platform independent and more importantly, is publically available and accepted. Table 6.6 shows the typical structure of a CWE Arti-

Table 6.5

Relevance of CWE Articles for Flagged Security Vulnerabilities

Vulnerability type	Recommended CWE Articles with the titles		TFIDF Score	Relevance
Cross site scripting vulnerability	CWE 079	Improper Neutralization of Input During Web Page Generation	1.0470	High
	CWE 644	Improper Neutralization of HTTP Headers for Scripting Syntax	0.4749	Medium
Hardcoded constant database password	CWE 522	Insufficiently Protected Credentials	0.7739	High
	CWE 259	Use of Hard-coded Password	0.7359	High
	CWE 256	Plaintext Storage of a Password	0.6858	High
	CWE 640	Weak Password Recovery Mechanism for Forgotten Password	0.6342	Medium
	CWE 798	Use of Hard-coded Credentials	0.5803	High
	CWE 261	Weak Cryptography for Passwords	0.5737	Medium
	CWE 620	Unverified Password Change	0.5231	Low
	CWE 309	Use of Password System for Primary Authentication	0.5142	Low
	CWE 187	Partial Comparison	0.5003	Low
	CWE 260	Password in Configuration File	0.4536	Medium
	CWE 258	Empty Password in Configuration File	0.3816	Medium
	CWE 263	Password Aging with Long Expiration	0.3794	Low
HTTP Response splitting vulnerability	CWE 113	Improper Neutralization of CRLF Sequences in HTTP Headers	1.5627	High
	CWE 650	Trusting HTTP Permission Methods on the Server Side	0.6569	High
	CWE 644	Improper Neutralization of HTTP Headers for Scripting Syntax	0.5360	High
Non-constant SQL string passed to execute method	CWE 089	Improper Neutralization of Special Elements used in SQL stmt.	0.4768	High
	CWE 484	Omitted Break Statement in Switch	0.3839	Low

cle. Important sections of a CWE article, to be used as training resource in the proposed recommender system, are highlighted in Table 6.6.

Table 6.6

Structure of a Typical CWE Article

Description
Time of Introduction
Applicable Platforms
Modes of Introduction
Common Consequences
Likelihood of Exploit
Enabling Factors of Exploitation
Detection Methods
Demonstrative Examples
Observed Examples
Potential Mitigations
Relationships
Relationship Notes
Taxonomy Mappings
Related Attack Patterns
White Box Definitions
References
Content History

Although the Potential Mitigations section will serve as the core of training materials, other sections like Description, Demonstrative Examples, Detection Methods, and Common Consequences can also be useful for developers.

6.3 Conclusions

In this chapter, we demonstrated the technical feasibility of mapping vulnerability type descriptions to a large collection of articles describing solutions, hence validating the pro-

posed architecture of the system. We were able to map most of the flagged vulnerabilities to articles in CWE repository. Also, we found that most of the articles in CWE repository contain mitigation strategies and other useful information to understand and avoid security vulnerabilities. Hence, such articles can be used to train the software developers.

Given the technically feasible architecture, we conducted human subject evaluation to compare the effectivness of the proposed system to a commercial tools that provide similar recommendations to software developers. The details about human subject evaluation are given in next chapter.

CHAPTER 7

HUMAN SUBJECT EVALUATION

The human subject evaluation of the recommender system is a continuation to the work presented in chapter 6. This chapter outlines how the study was designed and executed.

In this chapter, we answer the following research questions:

1. How do the recommendations based on CWE repository compare to recommendation from commercial tool's repository in terms of training suitability?
2. How do the tf–idf and Jaccard index algorithms compare to each other to identify relevant articles in public repositories given the evaluation of human subjects?

7.1 Apparatus, Materials, and Artifacts

Jaccard index and tf–idf algorithms produce a numeric score for each possible pair of CWE article and identified vulnerability; however, for evaluation, we selected the top two most relevant articles using each algorithm for each vulnerability.

In addition to using Jaccard index and tf–idf based approaches, we also used a commercial tool to analyze our target source code. The primary purpose of using a commercial tool was to use it as a standard to evaluate our recommender system. The commercial tool uses a proprietary repository to recommend articles for training the software developers. It fetches exactly one article for each flagged security vulnerability.

Hence, five articles in total were provided to each human expert for evaluation. The articles were arranged in random order. Any identifying information about the article (e.g., whether it has been taken from CWE or commercial tools repository) is removed before it is presented to the human expert for evaluation.

Table 7.1 summarizes the identified vulnerabilities and the recommended articles for each vulnerability.

7.2 Experimental Design

The evaluation of each article was based on three criteria: relevance, suitability for training, and the amount of information in the article. We used a five point Likert scale for evaluation. For relevance and suitability $1 = Not\ relevant$ or $Not\ suitable$ and $5 = Highly\ relevant$ or $Highly\ suitable$, whereas for amount of information $1 = Too\ little$, $3 = Just\ right$, and $5 = Too\ much$ information.

There were 44 human subjects who voluntarily participated in the study. Each participant was assigned to evaluate the articles for one vulnerability. All subjects had previous software development experience and had sound software security background. Table 7.2 summarizes the information about participants, including major field of study.

7.3 Data Collection

The participants were briefed about the recommender system and the task to be completed. They were allotted one hour to complete the survey. The subject experts reviewed the survey material to make sure that participants had a sufficient amount of time to carefully read and evaluate the survey material. Any identifying information about participants

Table 7.1

Summary of Recommended Articles

Vulnerability	Description	Number of recommended articles			Total articles for human subject to evaluate
		tf–idf	Jaccard index	Commercial tool	
V1	XSS - Cross Site Scripting vulnerability	2	2	1	5
V2	SQL Injection vulnerability	2	2	1	5
V3	HTTP Response splitting vulnerability	2	2	1	5
V4	Hardcoded constant database password	2	2	1	5

Table 7.2

Summary of Participants

No. of participants			**Average experience**	
Computer Science	Software Engineering	Electrical & Computer Engineering	Software development (years)	Software security (years)
25	13	6	1.68	0.6

was detached from the collected data before it was compiled. The researchers obtained formal permission from Institutional Review Board (IRB)[1] of the Mississippi State University before conducting the human subject evaluation. More details about data collection and experiment design can be found in Appendix B.

7.4 Analysis Methods

For analysis of data we used paired t–test and mixed model ANOVA. Since tf–idf and Jaccard index are two different methods of measurements therefore the paired t–test[2] is the suitable way to compare their means. Using the same test we compared the means of tf–idf and Jaccard index to the mean of commercial tool.

Mixed model ANOVA[3] is suitable as we are testing for differences among independent groups and subjecting our participants to repeated measures of independent variables. The groups in our data are formed on the basis of four vulnerabilities (serving as between–group factor) and three approaches (serving as within–group factor).

Details about the paired t–test and ANOVA can be found in the following section.

7.5 Results

This section discusses the results of the human subject evaluation. As per our study design, each participant was asked to evaluate five articles identified by the three approaches (i.e., tf–idf, Jaccard Index, and commercial tool) for one vulnerability. The four vulnera-

[1]IRB, `http://orc.msstate.edu/humansubjects/`
[2]Paired t–test, `http://www.statstutor.ac.uk/resources/uploaded/paired-t-test.pdf/`
[3]Mixed Model ANOVA, `https://en.wikipedia.org/wiki/Mixed-design_analysis_of_variance/`

bilities (V_1, V_2, V_3, and V_4) served as a between–groups factor while the three approaches served as a within–groups factor in a 4x3 mixed model ANOVA. The analysis shows that both tf–idf and commercial tool were significantly better than Jaccard index based approach $F(2, 80) = 12.57, p < .001$.

For further analysis we excluded the Jaccard index based approach and conducted the ANOVA using four vulnerabilities and two approaches (i.e., tf–idf and commercial tool) in a 4x2 mixed model. The small $F - value$ of $0.006(p < .85)$ strongly suggests that both approaches are equally good.

The following subsections discuss results about 'suitability for training' and 'relevance'.

7.5.1 Suitability for training

The estimated marginal mean for 'suitability for training' of different approaches, as evaluated by the human subjects, is shown in Figure 7.1. The result of paired t–test for 'suitability for training' can be found in Table 7.3. The results are explained below.

tf–idf vs. commercial tool. The paired t–test suggests that there is no significant difference between the tf–idf based approach and commercial tool in terms of finding articles suitable for training the software developers. Estimated marginal mean shown in Figure 7.1 and Figure 7.3 suggest the same. Hence, the articles from CWE repository are as good as the articles from commercial tool's repository in terms of suitability for training; Hence, the answer to our first research question.

Table 7.3

Paired Samples Test for Suitability

Pair	**Mean**	**Std. Dev.**	**Std. Error Mean**	**t**	**df**	**Sig. (2 tailed)**
tf–idf - Commercial tool	.045	1.461	.220	.206	43	.838
tf–idf - Jaccard index	.840	1.627	.245	3.427	43	.001
Commercial tool - Jaccard index	.795	1.935	.291	2.726	43	.009

tf–idf and commercial tool vs. Jaccard index. The paired t–test suggests that the tf–idf based approach and commercial tool both performed significantly better than the Jaccard index based approach in terms of suitability of training. The estimated marginal mean in Figure 7.3 suggest the same.

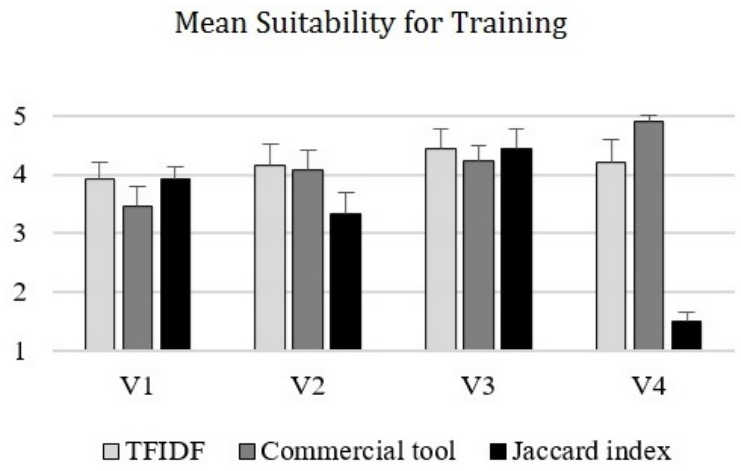


Figure 7.1

Mean Suitability for Training

7.5.2 Relevance

The relevance of the identified articles to the detected vulnerabilities, as evaluated by the human subjects, is shown in Figure 7.2 and Figure 7.3. The result of paired t–test for relevance can be found in Table 7.4. The results are explained below.

tf–idf vs. commercial tool. The paired t–test for relevance suggests that there is no significant difference between the tf–idf based approach and commercial tool in terms of finding articles that are relevant to the flagged vulnerabilities. The estimated marginal

Table 7.4

Paired Samples Test for Relevance

Pair	**Mean**	**Std. Dev.**	**Std. Error Mean**	**t**	**df**	**Sig. (2 tailed)**
tf–idf - Commercial tool	-.113	1.125	.169	-.670	43	.506
tf–idf - Jaccard index	.500	1.210	.182	2.740	43	.009
Commercial tool - Jaccard index	.613	1.204	.181	3.378	43	.002

mean for relevance also in Figure 7.2 and Figure 7.3 suggest the same. Hence, the articles from the CWE repository are as good as the articles from commercial tool's repository in terms of their relevance to the flagged vulnerabilities.

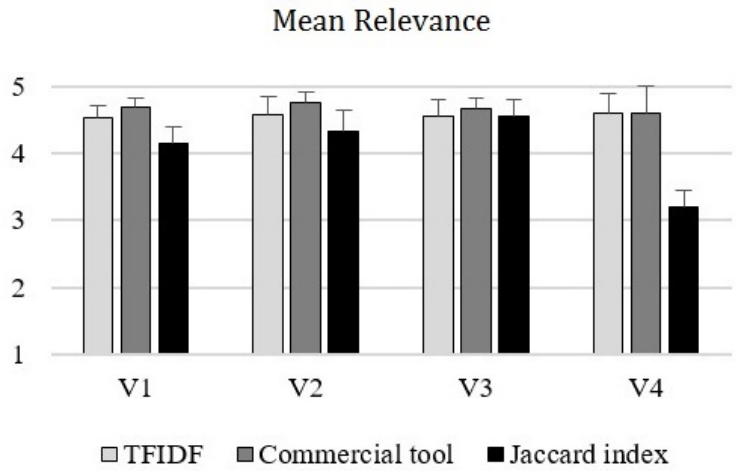


Figure 7.2

Mean Relevance

tf–idf vs. Jaccard index. The paired t–test for relevance suggests that the tf–idf based approach is significantly better than the Jaccard index based approach. The estimated marginal mean in Figure 7.3 suggest the same. This give us basis for answering our second research question.

In this chapter, with the help of human subject evaluation, we concluded that CWE repository is as effective as commercial tool's repsitory in terms of 'suitability of training'. Moreover, we found that tf–idf based approach as accurate as commercial tool in terms of finding articles relevant to flagged vulnerabilities.

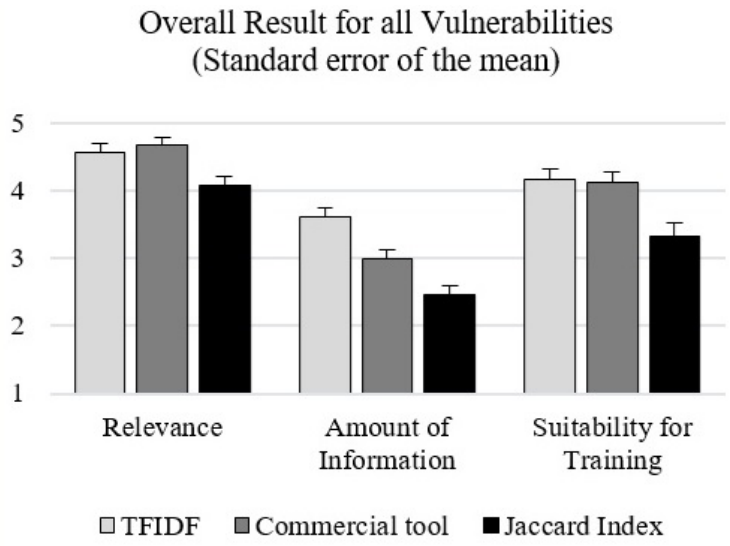


Figure 7.3

Overall result for all vulnerabilities

In the next chapter, we discuss the tailored version of tf–idf and Jaccard index based approaches and experiments conducted to evaluate their performance.

CHAPTER 8

TAILORED ALGORITHMS

This chapter discuss two important experiments. In the first part, i.e., section 8.1, we discuss the the design and evaluation of tailored version of our tf–idf and Jaccard index based approaches to map discovered vulnerabilities to the articles in public vulnerability repository. In the second part, i.e., section 8.2, we analyze security vulnerability trends in open source software.

8.1 Evaluation of Tailored Approaches

Improving the accuracy of information retrieval algorithms has always been challenging. In chapter 7, we discussed how stemming and stop words can help improving the accuracy of our approaches. In addition to stemming and stop words, our tailored approaches also utilize the Part of Speech (POS) Tagger API and custom built tools to transform the entire dataset. The similarity scores are calculated after the transformation. In the future, we also plan to use parameters like length of articles in re-ranking the recommended training articles.

8.1.1 POS Tagger and Noun–based Search

A Part-Of-Speech Tagger (POS Tagger) [59] is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine–grained POS tags like 'noun-plural'.

Literature suggests that taking the linguistic narture of the words into account can be another way to improve the accuracy of information retrieval approaches. A study on Noun–based indexing of software artifacts [12] reported significant improvement in the accuracy of information retrieval. We are introducing this idea in our recommender system.

8.1.2 Extracting Nouns

The dataset used in our experiment, (i.e., flagged vulnerabilities and training articles from public vulnerability repository), are pre–processed using POS Tagger and the custom built Noun Extractor tool. The working is explained in Figure 8.1.

The POS Tagger creates a 'tagged–file' for each input file. The 'tagged–file' is then input to the cutom built Noun extractor tool, which extracts all nouns and stores them into a new file. Hence the entire dataset is transformed. Our tailored approach modifies the working of the recommender module, The 'Modified Recommender Module' is shown in Figure 8.2.

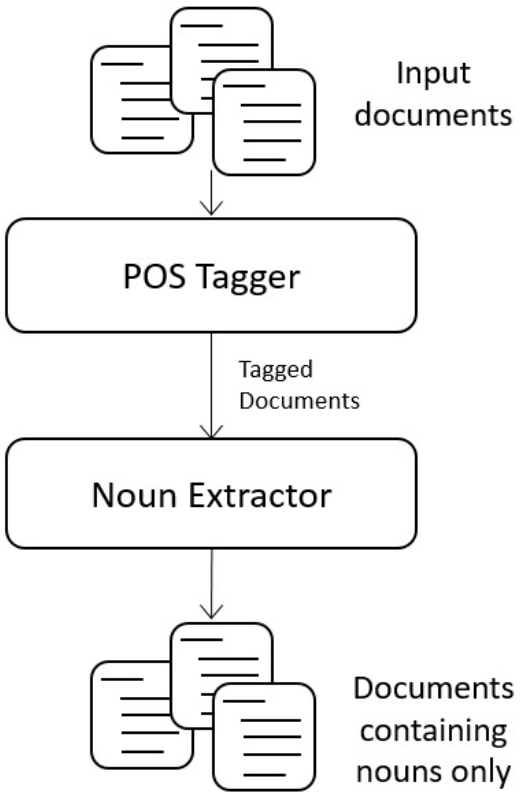


Figure 8.1

Extracting nouns using POS Tagger

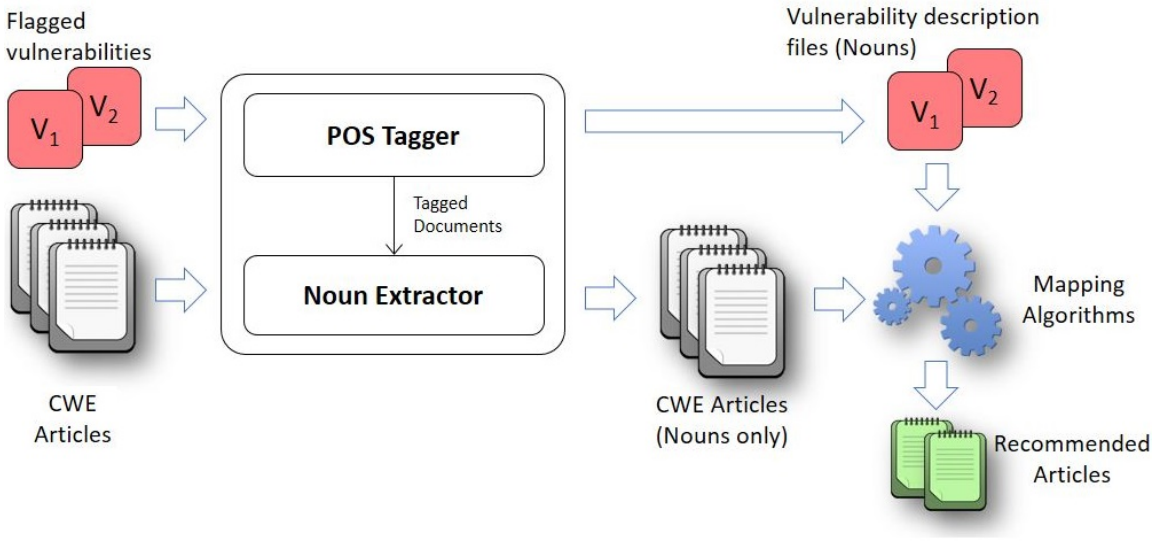


Figure 8.2

The Modified Recommender Module

8.1.3 Results

In this section we compare the primitive approaches, discussed in chapter 7, with our tailored approaches. We use Receiver Operating Characteristic (ROC)[1] curve to compare the ranking performance of primitive and tailored tf–idf and Jaccard index based approaches.

8.1.3.1 Primitive vs. Tailored tf–idf Based Approach

The comparison between the primitive and tailored tf–idf based approach is shown in Figure 8.3. The smaller values represent better ranking. For a given vulnerability, say Cross Site Scripting or V_1, the tailored approach was able to rank all the relevant articles in a better way compared to the primitive tf–idf based approach.

[1]ROC Curve, `https://en.wikipedia.org/wiki/Receiver_operating_characteristic` (accessed May 15, 2016).

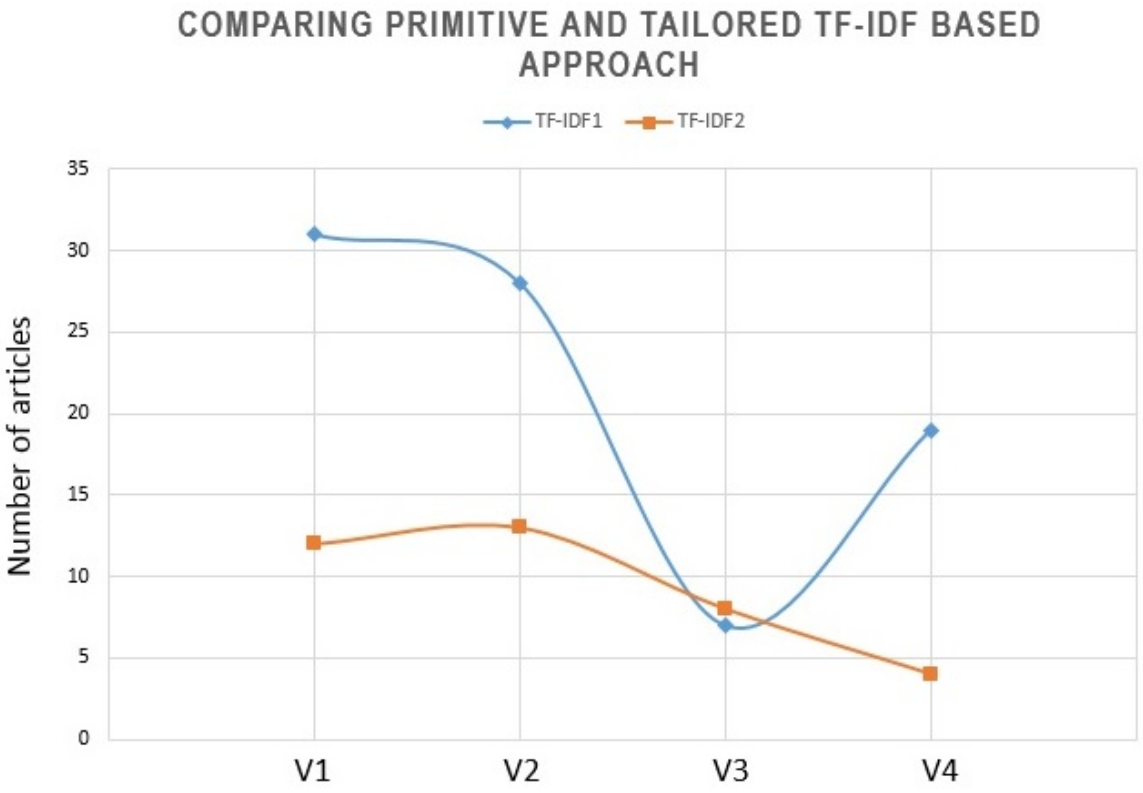


Figure 8.3

Primitive vs. tailored tf–idf based approach

The tailored approach ranked all relevant articles within top 12 articles compared to the primitive tf–idf based approach which ranked the same articles within top 31. Same is true for V_2 and V_4; However, for the HTTP Response Splitting vulnerability, or V_3, both approaches performed equally good.

The ranking data for primitive and tailored tf–idf based approaches is given in Table 8.1.

Table 8.1

Primitive vs. tailored tf–idf based approach

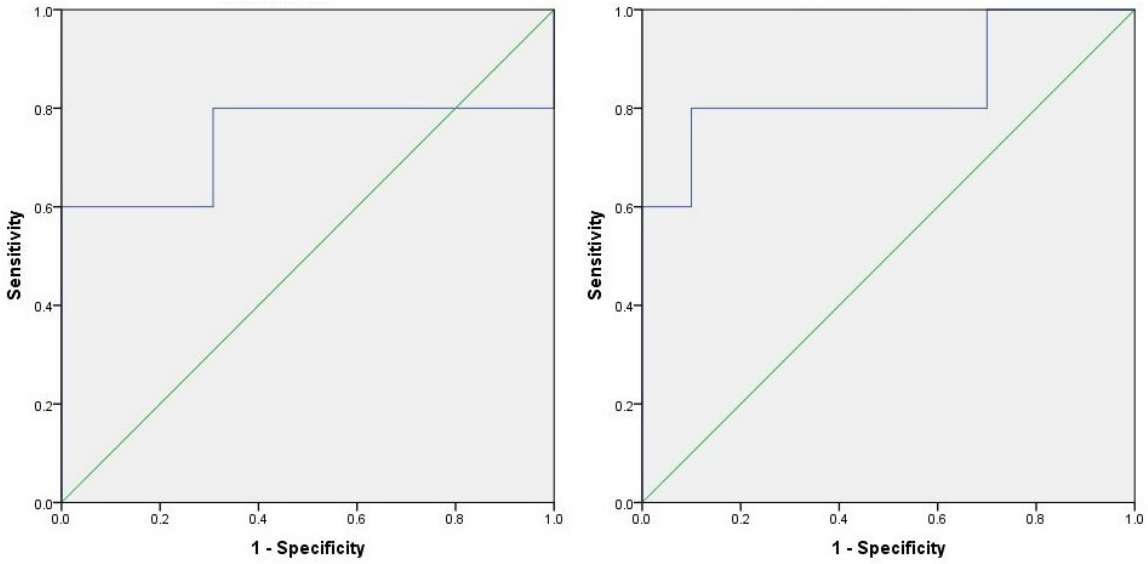
Vulnerability type	Top 'n' articles that contain all relevant articles	
	Primitive tf–idf based approach	Tailored tf–idf based approach
V_1, Cross Site Scripting	31	12
V_2, Hard–coded database password	28	13
V_3, HTTP Response splitting	7	8
V_4, SQL injection	19	4

The ROC curves for primitive and tailored tf–idf based approach for Cross Site Scripting vulnerability i.e., V_1, are shown in Figure 8.4.

The ROC curves for primitive and tailored tf–idf based approach for Hard–coded constant database password i.e., V_2, are shown in Figure 8.5.

The ROC curves for primitive and tailored tf–idf based approach for HTTP Response Splitting vulnerability i.e., V_3, are shown in Figure 8.6.

The ROC curves for primitive and tailored tf–idf based approach for SQL injection vulnerability i.e., V_4, are shown in Figure 8.7.



Area Under the Curve

Test Result Variable(s): TFIDF1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.738	.175	.096	.395	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Area Under the Curve

Test Result Variable(s): TFIDF2_Score

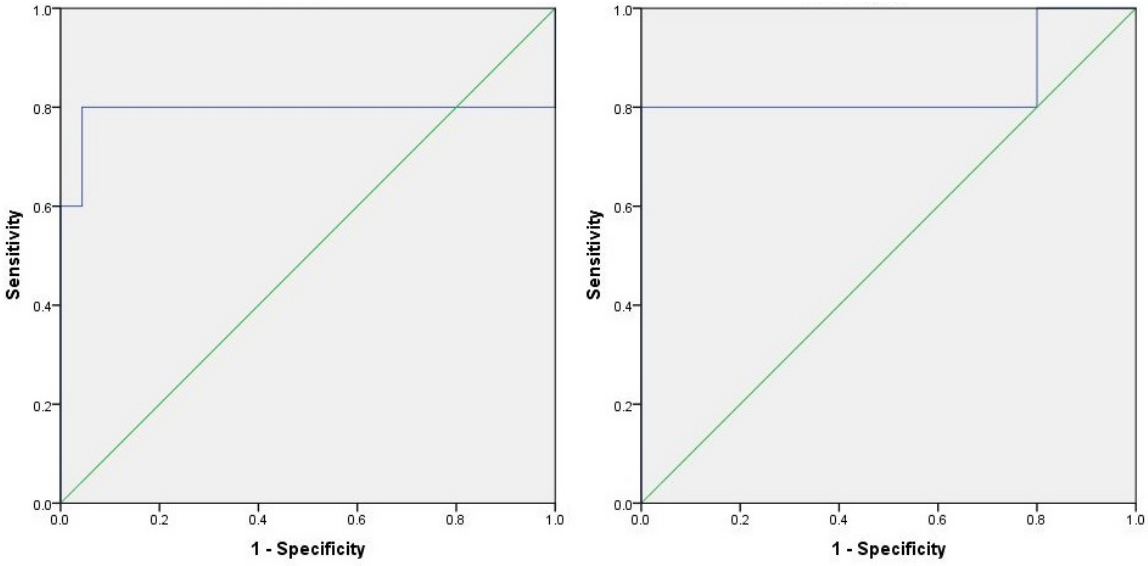
Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.840	.131	.037	.583	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.4

ROC curves for primitive (L) and tailored (R) tf–idf based approaches for V_1



Area Under the Curve

Test Result Variable(s): TFIDF1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.791	.177	.044	.443	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Area Under the Curve

Test Result Variable(s): TFIDF2_Score

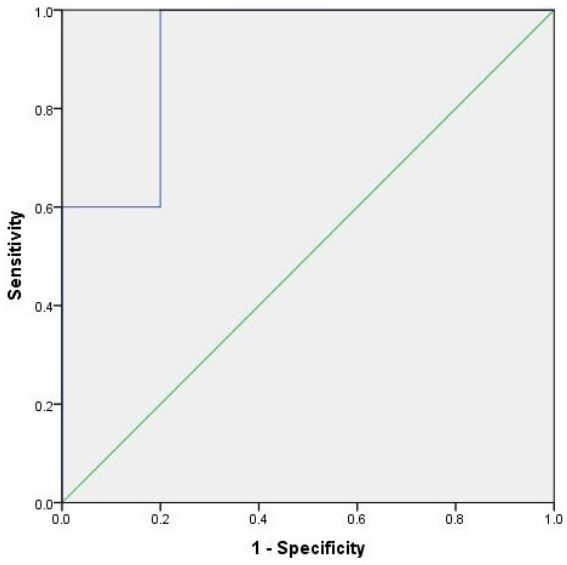
Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.840	.147	.037	.552	1.000

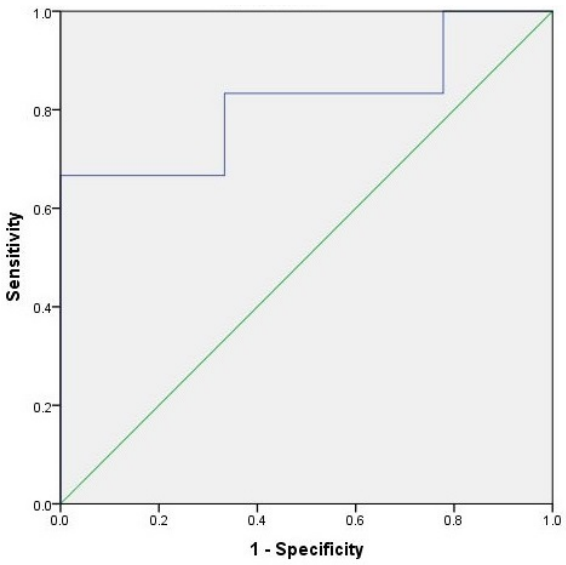
a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.5

ROC curves for primitive (L) and tailored (R) tf–idf based approaches for V_2





Area Under the Curve

Test Result Variable(s): TFIDF1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.920	.072	.010	.778	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Area Under the Curve

Test Result Variable(s): TFIDF2_Score

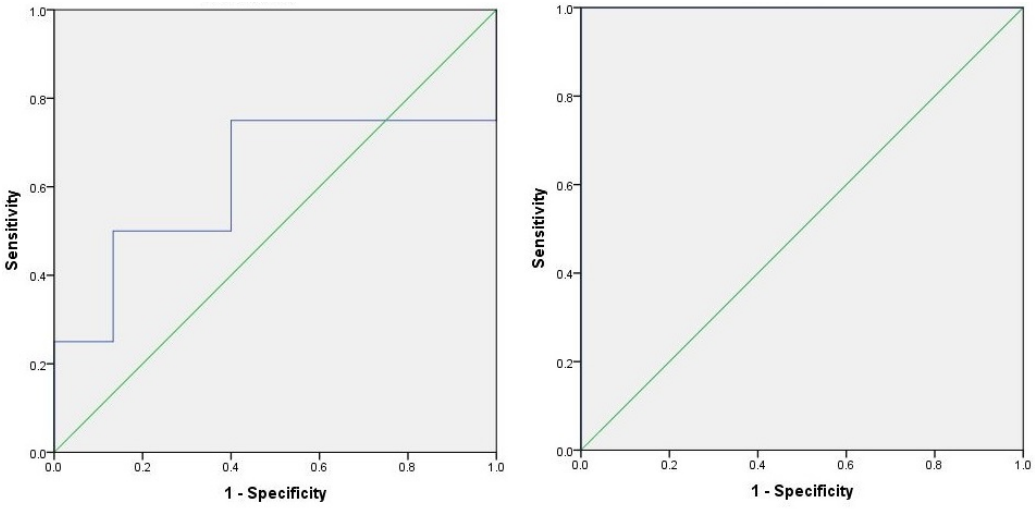
Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.815	.130	.045	.561	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.6

ROC curves for primitive (L) and tailored (R) tf–idf based approaches for V_3



Area Under the Curve

Test Result Variable(s): TFIDF1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.617	.200	.484	.225	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Area Under the Curve

Test Result Variable(s): TFIDF2_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
1.000	.000	.004	1.000	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.7

ROC curves for primitive (L) and tailored (R) tf–idf based approaches for V_4

8.1.3.2 Primitive vs. Tailored Jaccard Index Based Approach

The comparison between the primitive and tailored Jaccard index based approach is shown in Figure 8.8. As discussed in the previous section, the smaller scores represent better ranking. The tailored version of Jaccard index based approach performed better than the primitive approach for Cross Site Scripting vulnerability (V_1), the hard–coded password vulnerability (V_2), and the SQL injection vulnerability (V_4). However, due to some outliers, the tailored approach performed worse than the primitive approach for HTTP Response Splitting vulnerability (V_3).

The ranking data for primitive and tailored Jaccard index based approaches is given in Table 8.2.

Table 8.2

Primitive vs. tailored Jaccard index based approach

Vulnerability type	Top 'n' articles that contain all relevant articles	
	Primitive Jaccard index based approach	Tailored Jaccard index based approach
V_1, Cross Site Scripting	59	9
V_2, Hard–coded database password	32	18
V_3, HTTP Response splitting	24	68
V_4, SQL injection	31	11

For further analysis, we removed the outliers from the ranking data, the modified results are shown in Figure 8.9.

The ROC curves for primitive and tailored Jaccard index based approach for Cross Site Scripting vulnerability i.e., V_1, are shown in Figure 8.10.

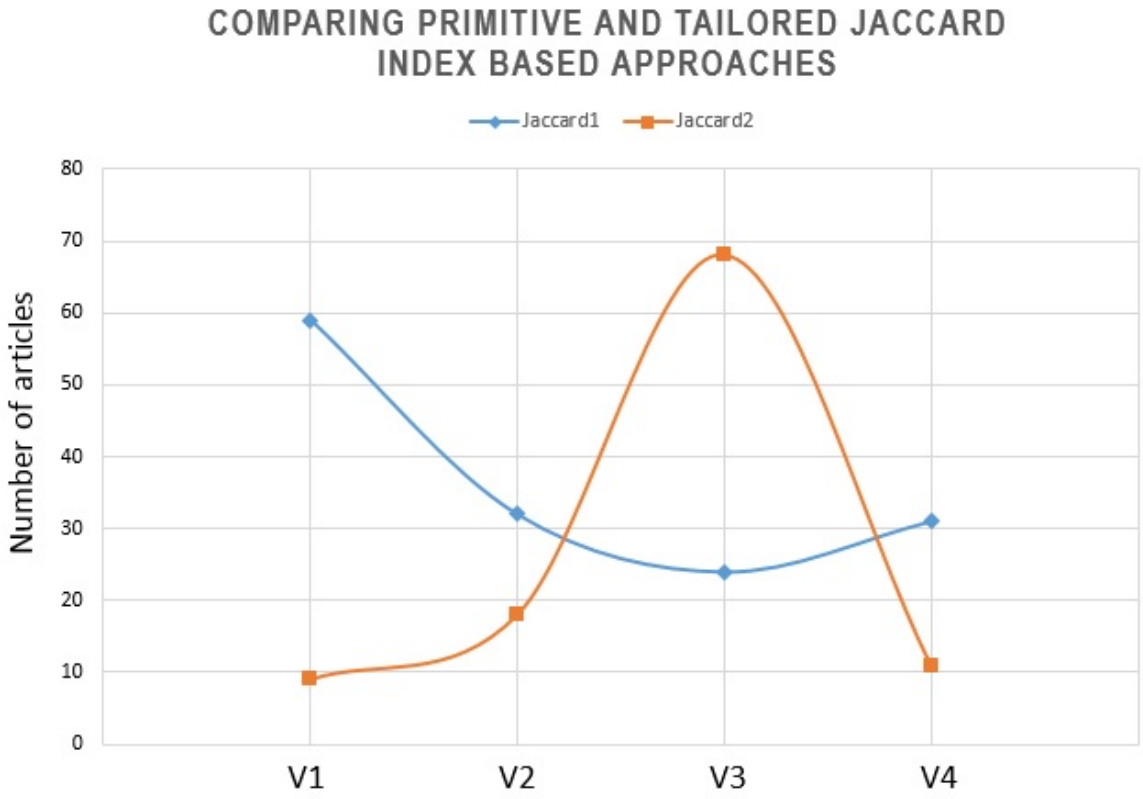


Figure 8.8

Primitive vs. tailored Jaccard index based approach

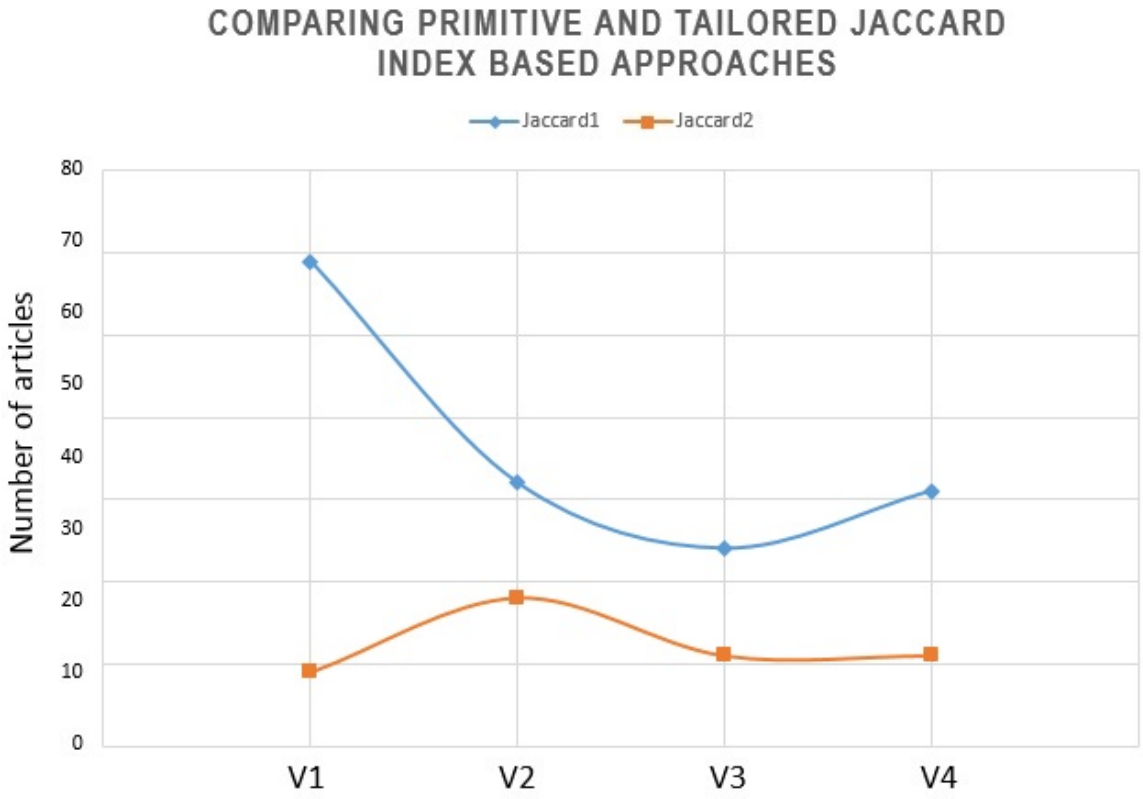
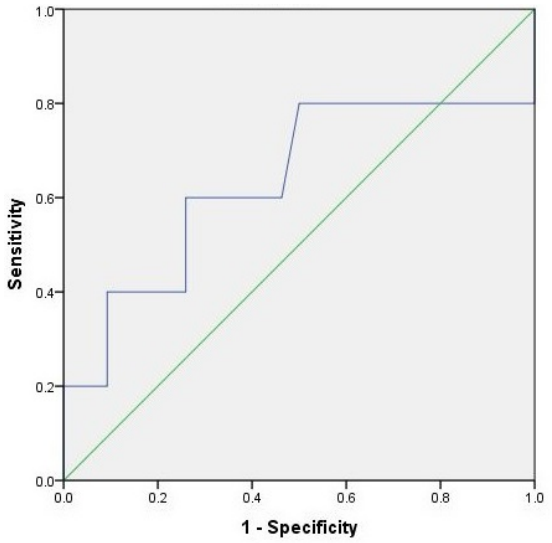


Figure 8.9

Primitive vs. tailored Jaccard index based approach - outliers excluded



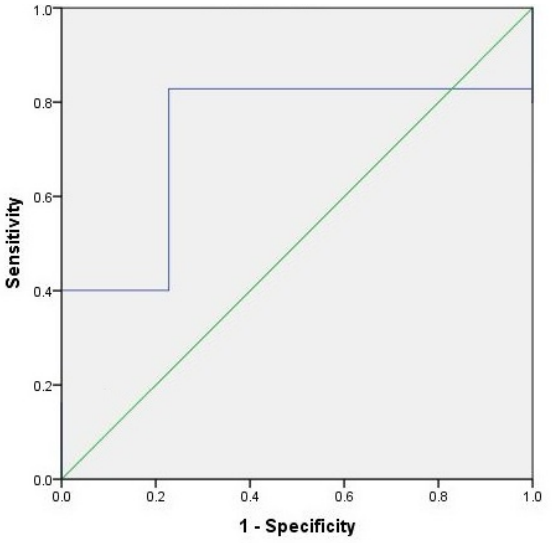
Area Under the Curve

Test Result Variable(s): Jaccard1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.633	.163	.327	.315	.952

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5



Area Under the Curve

Test Result Variable(s): Jaccard2_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.705	.170	.162	.372	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.10

ROC curves for primitive (L) and tailored (R) Jaccard index based approaches for V_1

The ROC curves for primitive and tailored Jaccard index based approach for Hard–coded constant database password i.e., V_2, are shown in Figure 8.11.

The ROC curves for primitive and tailored Jaccard index based approach for HTTP Response Splitting vulnerability i.e., V_3, are shown in Figure 8.12.

The ROC curves for primitive and tailored Jaccard index based approach for SQL injection vulnerability i.e., V_4 is shown, are Figure 8.13.

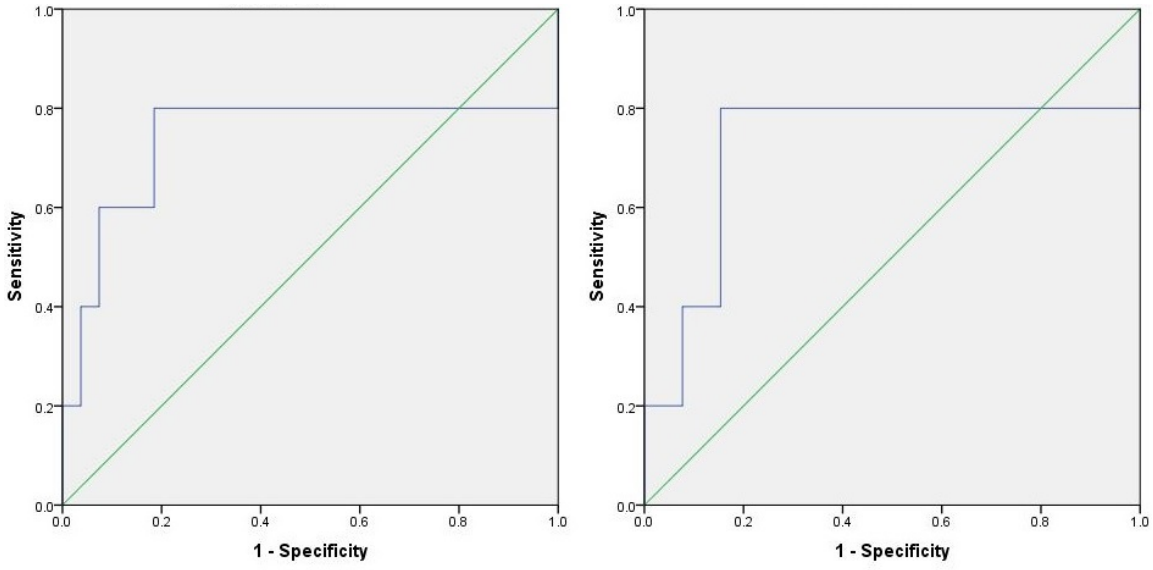
8.2 Analyzing Security Vulnerability Trends in Open Source Software

We analyzed the data regarding reported vulnerabilities[2],in different open source software systems. In case of Mozila Firefox, we analyzed data for Denial of Service (DoS), Code Execution, Overflow, Cross Site Scripting (XSS), Bypass some mechanism, and Information Gain vulnerabilities reported over several years.

8.2.1 Results

Figure 8.14 and Figure 8.15 show trends of these reported vulnerabilities. We found evidence that the vulnerabilities discovered in a specific version of such widely used open source software product were propogated to the newer versions of the prodcut despite the fact that software developers working on newer versions of the products had access to the reported vulnerabilities. Which brings us to the conclusion that providing narrow and focused security training to software developers may bar propogation of known vulnerabilities to the newer versions of software product.

[2]CVE Datasource, `http://www.cvedetails.com/` (accessed Apr 22, 2016).



Area Under the Curve

Test Result Variable(s): Jaccard1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.741	.171	.092	.406	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Area Under the Curve

Test Result Variable(s): Jaccard2_Score

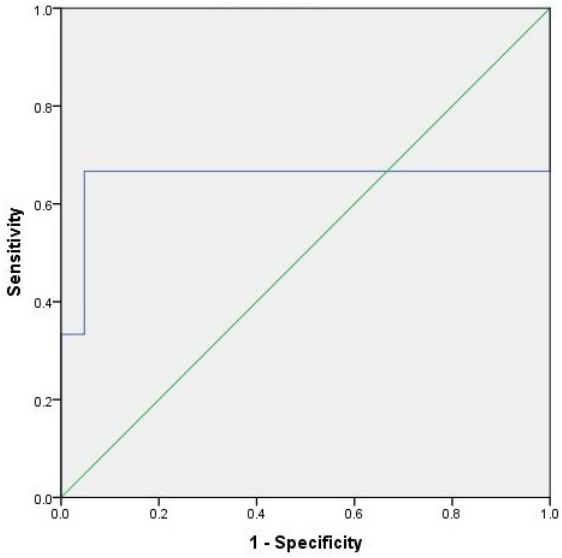
Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.723	.173	.153	.384	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.11

ROC curves for primitive (L) and tailored (R) Jaccard index based approaches for V_2



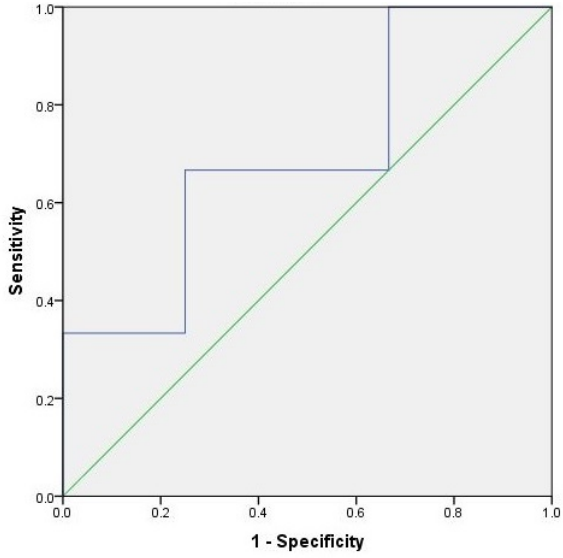
Area Under the Curve

Test Result Variable(s): Jaccard1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.651	.267	.407	.128	1.000

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5



Area Under the Curve

Test Result Variable(s): Jaccard2_Score

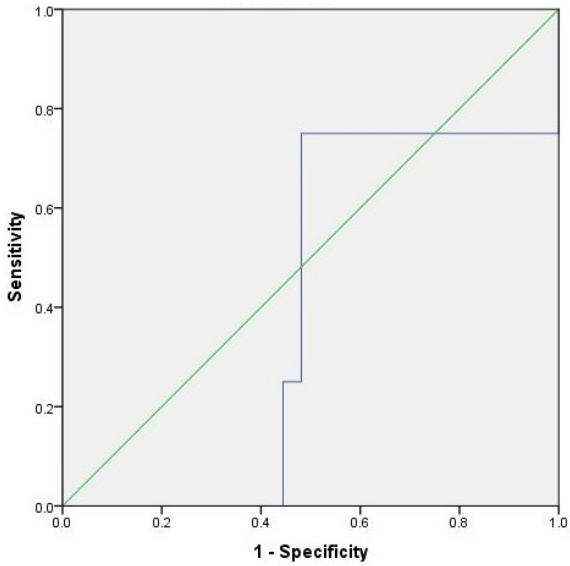
Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.694	.180	.312	.341	1.000

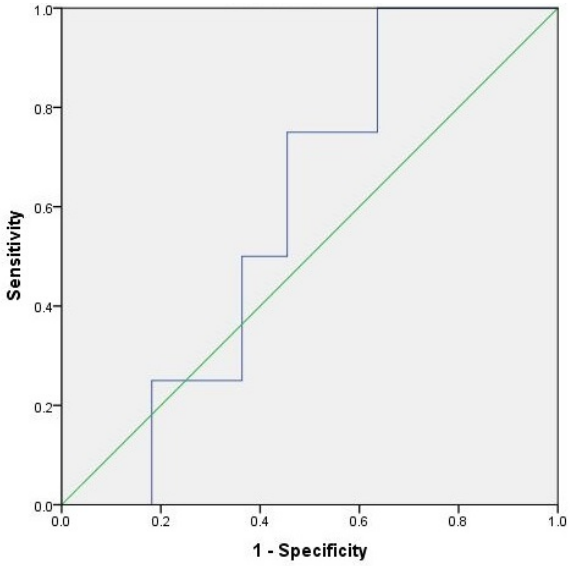
a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.12

ROC curves for primitive (L) and tailored (R) Jaccard index based approaches for V_3





Area Under the Curve

Test Result Variable(s): Jaccard1_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.398	.137	.517	.130	.667

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Area Under the Curve

Test Result Variable(s): Jaccard2_Score

Area	Std. Error[a]	Asymptotic Sig.[b]	Asymptotic 95% Confidence Interval	
			Lower Bound	Upper Bound
.591	.148	.602	.301	.881

a. Under the nonparametric assumption

b. Null hypothesis: true area = 0.5

Figure 8.13

ROC curves for primitive (L) and tailored (R) Jaccard index based approaches for V_4

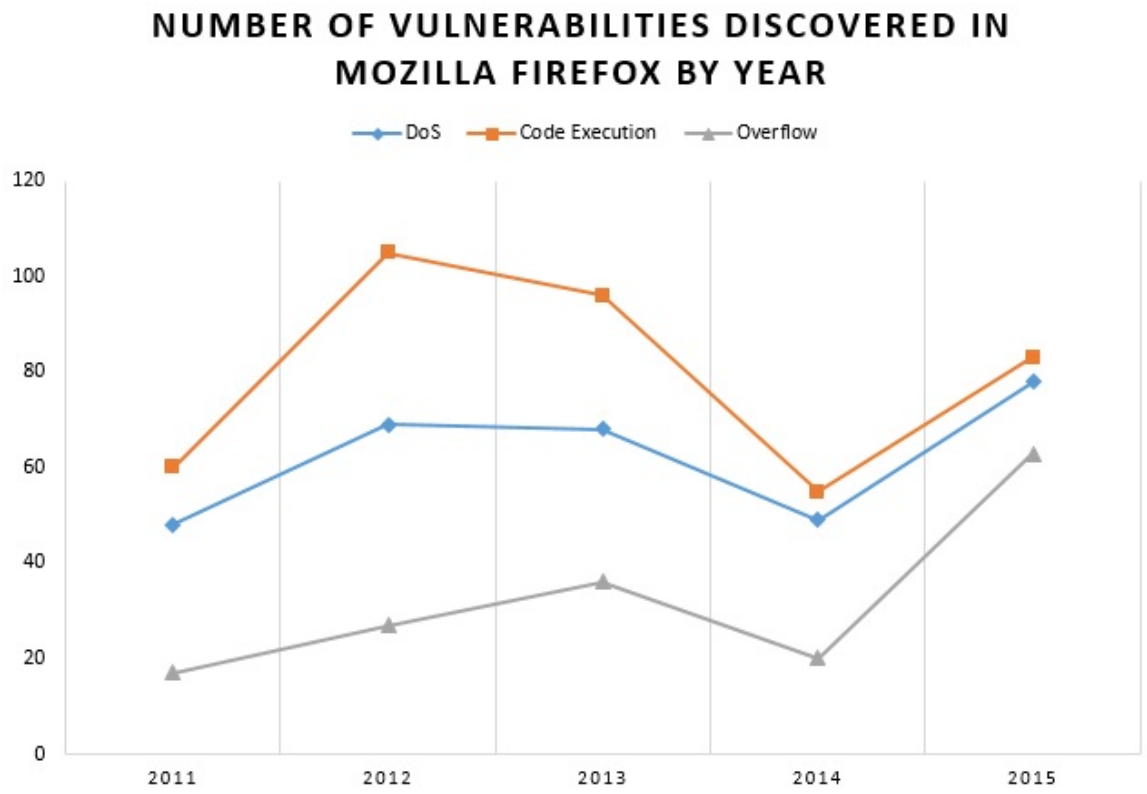


Figure 8.14

Reported Vulnerabilites for Mozilla FireFox (A)

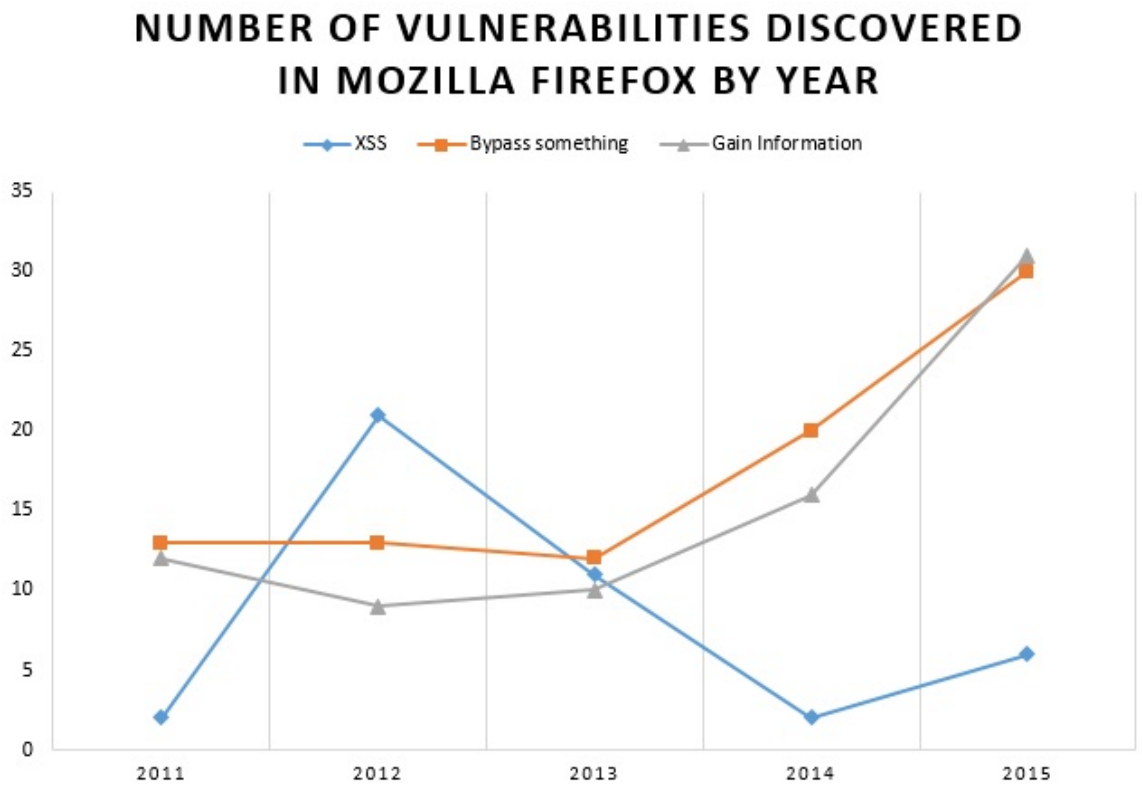


Figure 8.15

Reported Vulnerabilites for Mozilla FireFox (B)

CHAPTER 9

DISCUSSION

In this dissertation, with the help of experiments and human subject evaluation, we found that the potential vulnerabilities in an individual developer's source code can be automatically mapped to relevant articles in public vulnerability repositories that are suitable for training the developer regarding vulnerability mitigation.

This chapter discusses findings for our research questions.

9.1 Research Questions

In this section we lists the research questions, result for each question with evidence, and significance of result for our recommender system and rest of the research community.

1. How do the recommendations based on CWE repository compare to recommendation from HP Fortify in terms of training suitability?

   Result:
   Based on the results of human subject evaluation, given in chapter 7, it is concluded that articles in CWE repository are as suitable for training as articles in commercial tool's repository.

   Significance:
   CWE repository can be used as knowledgebase for recommender system discussed in this study and other similar systems. Moreover, automated code transformation tools may use the information in CWE repository

2. How do the tf–idf and Jaccard index algorithms compare to each other to identify relevant articles in public repositories given the evaluation of human subjects?

Result:
Based on the results of human subject evaluation, given in chapter 7, it is concluded that tf–idf based approach is significantly better than Jaccard index based approach in terms of identifying relevant articles from public vulnerability repository.

Significance:
Our future research will focus on tf–idf based approach; moreover, our findings will benefit the research community.

3. Regarding tailored approaches,

   (a) How does the tailored mapping algorithm based on tf–idf scores and additional attributes, i.e., recency and completeness of the articles compare, to the other algorithms in terms of relevance?

   Result:
   Based on the results of experiment, discussed in chapter 8, it is concluded that tailored tf–idf based approach performed better than primitive tf–idf based approache (discussed in chapter 7) in terms of ranking the articles.

   Significance:
   Modified architecture of our recommender system should have a customizable POS tagger module, which should allow extracting other parts of speech in addition to nouns.

   (b) How does the tailored mapping algorithm based on Jaccard index scores and additional attributes, i.e., recency and completeness of the articles, compare to the other algorithms in terms of relevance?

   Result:
   Based on the results of experiment, discussed in chapter 8, it is concluded that tailored Jaccard index based approach performed better than primitive Jaccard index based approache (discussed in chapter 7) in terms of ranking the articles. However, none of the Jaccard index based approach performed better than tf–idf based approaches.

   Significance:
   Future work of our research should use tf–idf based approaches as primary means of mapping flagged vulnerablities to training articles.

4. What is the statistical profile of developers in the case study based on the following-parameters?

   (a) Tendency to cause vulnerabilities over the case study window

   (b) Recency of induced vulnerabilities

(c) Severity of induced vulnerabilities

Result:
Based on the analysis of open source software systems, given in chapter 8, we found evidence that the vulnerabilities discovered in a specific version of a widely used open source software product can be propogated to the newer versions of the prodcut. Which brings us to the conclusion that narrow and focused security training to software developers may bar propogation of known vulnerabilities to the newer versions of software product.

Significance:
There is need to analyze factors that result in increase in vulnerabilities.

9.2 Threats to Validity

The experiments and results discussed in this dissertation have the following threats to validity.

9.2.1 Generic Limitation of Static Analysis Tools

The generic limitations to the static analysis tools may introduce some inaccuracies or limitations in our system. The static analysis tools may produce false positive or false negative results. A false positive refers to detection of a vulnerability by the tool in the given code which actually is not a vulnerability [43]. Often a high percentage of discovered vulnerabilities are actually false positives [4]. The results generated by such tools can cause unproductive consumption of time and resources [8].

9.2.1.1 Implication of False Positive Results

Since the recommender discussed in this study relies on the result of static analysis tool therefore some of the security–training recommended by our system may not actually be needed. On the other hand a false positive may still indicate an undesirable coding pattern

for which training would be helpful. However, the static analysis tool used in this study may not be representative of other similar tools.

9.2.1.2 Implication of False Negative Results

A false negative detection refers to a vulnerability that actually exists in a software code repository but the static analysis tools is unable to detect. False negative detections means our system will not recommend training to the software developers which should have been recommended.

9.2.2 Analysis Limited to one Open Source System

The experiments conducted in this dissertation are limited to the analysis of one open source system. We discussed in chapter 6 that the target system, i.e., Tolven 2.0, has already been used by research community in security related studies. However, Replicating the study to other open source systems, especially with known vulnerabilities, will help to further validate the effectiveness of the proposed recommender system.

9.2.3 Analysis Limited to one Open Source System

We compare the articles identified by our recommender system to only one commercial tool's recommended articles. Comparing the proposed system with other similar systems will also help to validate the effectiveness and accuracy of our system.

9.2.4 Human Factors

Some of our findings solely rely on the human subject evaluation; e.g., we found that the articles in CWE repository are as suitable for training as the articles in the commercial

tool's repository. Similarly, with the help of human subject evaluation we found that tf–idf based approach performed better than the Jaccard index based approach. The human bias may be a threat to the validity of our findings.

In this chapter, we presented the result and its significance for each research question. We also discussed different factors that impose threats to the validity of our results. The next chapter focuses on contribution and future work of this dissertation.

CHAPTER 10

CONCLUSIONS

10.1 Hypothesis

Before moving to the conclusion, let's revisit the hypothesis for our research:

> Potential vulnerabilities in an individual developer's source code can be automatically mapped to relevant articles in public vulnerability repositories that are suitable for training the developer regarding vulnerability mitigation.

10.2 Conclusion

In this study, with the help of human subject evaluation, we found that the proposed computer security–training recommender (with tf–idf based approach) performed as good as the commercial tool in terms of finding relevant articles from their respective repsitories based on the flagged vulnerabilities. We also found that both the CWE and commercial tool's repository host articles which are equally suitable for training the software developers. Hence, the proposed recommender system may be adopted by the software organizations as a low-cost alternative to commercial tools. Training articles from public vulnerability repository are not limited on any specific programming language or platform which gives our system an edge over the commercial tool.

10.3 Contributions

This research has the following contributions.

10.3.1 The proposed architecture

The major contribution of this research is the architectue of the system, which has been discussed in detail in chapter 5. The proposed architecture is flexible and can be used with a variety of open source tools, hence making it useful for software developers with different programming backgrounds.

10.3.2 Evaluation of Different Approaches for Recommending System

Evaluation of effectiveness of three approaches to retrieve training articles from public vulnerability repository based on the flagged vulnerabilities in the source code.

10.3.3 Proposing and Evaluating Tailored Approach

In addition to using the primitive tf–idf and Jaccard index based approach to retrieve training articles, this research proposed and evaluated tailored approaches which proved to be better than the primitive alogrithms.

10.3.4 Evaluation of CWE Articles for Suitability for Training

Empirical evidence, based on human subject evaluation, indicating that an open source tools based approach, which utilizes public vulnerability repository, can be as effective as commercially available tools for training the software developers.

10.3.5 Low Cost Alternative to Commercial Tools

The proposed system, which primarily relies on open source static analysis tools and public vulnerability repository is a potential alternative to the commercial tools. Software organizations may significantly reduce the training cost by utilizing the proposed system.

Also, the human subject evaluation has suggested that there is no significant difference between the proposed system and commercial tool.

10.4 Publication Plan

Parts of this research have already been published in different conference proceedings. However, we plan to publish the following completed work:

- A systemic mapping study on recommender systems for software engineering
- ComputerSecurity Training Recommender for software developers: Using POS-Tagger to pre-process dataset
- Analyzing security vulnerability trends in open source software systems

The identified target journals include IEEE Transaction on Software Engineering and Journal of Empirical Software Engineering.

10.5 Future Work

In work done so far, we presented the architecture for a computer–security training recommender and demonstrated the technical feasibility of mapping vulnerability type descriptions to a large collection of articles describing solutions. This work may be extended in the following possible ways:

10.5.1 Expanding Knowledgebase

The proof-of-concept implementation uses the CWE articles, however the other vulnerability databases e.g., National Vulnerability Database, NVD, host useful data related to security checklists, security related software flaws, misconfigurations, and impact metrics.

By utilizing the NVD database along with CWE articles, the scope of our recommender system may expand.

10.5.2 Improving Static Analysis

The static analysis tools produce false positive results. Using other static code analysis tools, in addition to FindBugs, and comparing their output may help reducing the false positive detections.

10.5.3 Automatic Code Generation

Although the primary objective of the recommender system is to recommend precise and focused solution to the security problems detected in developers code, there are other ways the proposed system may be used. For instance if the mitigation strategies given in the vulnerability repositories are detailed and standardized, a new module may be added in the recommender system which will replace the vulnerable code with secure code, hence adding a self–healing capability to the software. A Recommender system may periodically check for the newer solution and change the obsolete fixes with new ones.

10.5.4 Usability Studies

The usability study involves building a working prototype of the proposed recommender system and deploying it in software organizations to train the software developers and observing the improvement in terms of writing secure and robust code.

10.5.5 Prioritizing the Training for Software Developers

The severity level of the identified vulnerability may also be used in our system to prioritize the training for software developers. Common Vulnerability Scoring System (CVSS)[1] is one of the potential options that could serve our purpose.

10.5.6 Improving Mapping Algorithms

Different variations of mapping algorithms can designed and implemented to get further improvement in identifying relevant articles.

[1]CVSS, https://nvd.nist.gov/cvss.cfm/

REFERENCES

[1] T. Akinaga, N. Ohsugi, M. Tsunoda, T. Kakimoto, A. Monden, and K. I. Matsumoto, "Recommendation of software technologies based on collaborative filtering," *Proceedings: 12th Asia-Pacific Software Engineering Conference*, Taipei, Taiwan, Dec. 2005, IEEE Computer Society, pp. 209–214.

[2] B. Antunes, J. Cordeiro, and P. Gomes, "An approach to context-based recommendation in software development," *Proceedings: 6th ACM Conference on Recommender Systems*, Dublin, Ireland, Sept. 2012, ACM, pp. 171–178.

[3] B. Ashok, J. Joy, H. Liang, S. K. R. G. Srinivasa, and V. Vangala, "DebugAdvisor: a recommender system for debugging," *Proceedings: 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, Amsterdam, Netherlands, Aug. 2009, ACM, pp. 373–382.

[4] A. Austin and L. Williams, "One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques," *Proceedings: 5th International Symposium on Empirical Software Engineering and Measurement*, Banff, Canada, Sept. 2011, IEEE Computer Society, pp. 97–106.

[5] A. Azaria, A. Hassidim, S. Kraus, A. Eshkol, O. Weintraub, and I. Netanely, "Movie recommender system for profit maximization," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 121–128.

[6] E. A. Barbosa, A. Garcia, and M. Mezini, "A Recommendation System for Exception Handling Code," *Proceedings: 5th International Workshop on Exception Handling*, Zurich, Switzerland, June 2012, IEEE Computer Society, pp. 52–54.

[7] F. Belém, R. Santos, J. Almeida, and M. Gonçalves, "Topic diversity in tag recommendation," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 141–148.

[8] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: using static Analysis to find Bugs in the Real World," *Communications of the ACM*, vol. 53, no. 2, Feb. 2010, pp. 66–75.

[9] D. Binkley and D. Lawrie, *Encyclopedia of Software Engineering*, chapter Information retrieval applications in software development, John Wiley and Sons, Inc., Hoboken, NJ, USA, 2010, pp. 231–242.

[10] M. B. Blake, I. Saleh, Y. Wei, I. D. Schlesinger, A. Yale-Loehr, and X. Liu, "Shared service recommendations from requirement specifications: A hybrid syntactic and semantic toolkit," *Information and Software Technology*, May 2014, pp. 1–13.

[11] A. Böckmann, *MARS: Modular Architecture Recommendation System*, master's thesis, Faculty of Informatics of the University of Lugano, Ticino, June 2010.

[12] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving IR–based Traceability Recovery via Noun–based Indexing of Software Artifacts," *Journal of Software: Evolution and Process*, vol. 25, no. 7, July 2013, pp. 743–762.

[13] C. Castro-Herrera and J. Cleland-Huang, "Utilizing recommender systems to support software requirements elicitation," *Proceedings: 2nd International Workshop on Recommendation Systems for Software Engineering*, Cape Town, South Africa, May 2010, ACM, pp. 6–10.

[14] C. Castro-Herrera, J. Cleland-Huang, and B. Mobasher, "Enhancing Stakeholder Profiles to Improve Recommendations in Online Requirements Elicitation," *Proceedings: 17th IEEE International Requirements Engineering Conference*, Atlanta, GA, USA, Sept. 2009, IEEE Computer Society, pp. 37–46.

[15] R. Colomo-Palacios, I. González-Carrasco, J. L. López-Cuadrado, and Á. García-Crespo, "ReSySTER: A hybrid recommender system for Scrum team roles based on fuzzy and rough sets," *International Journal of Applied Mathematics and Computer Science*, vol. 22, no. 4, 2012, pp. 801–816.

[16] J. Cordeiro, B. Antunes, and P. Gomes, "Context-Based Recommendation to Support Problem Solving in Software Development," *Proceedings: Third International Workshop on Recommendation Systems for Software Engineering*, Zurich, Switzerland, June 2012, IEEE Computer Society, pp. 85–89.

[17] D. Čubranić and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," *Proceedings: 25th International Conference on Software Engineering*, Portland, OR, USA, May 2003, IEEE Computer Society, pp. 408–418.

[18] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: a project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, 2005, pp. 446–465.

[19] A. Danylenko and W. Löwe, "Context-aware recommender systems for non-functional requirements," *Proceedings: 3rd International Workshop on Recommendation Systems for Software Engineering*, Zurich, Switzerland, June 2012, IEEE Computer Society, pp. 80–84.

[20] E. Duala-Ekoko and M. P. Robillard, "Using structure-based recommendations to facilitate discoverability in APIs," *ECOOP 2011 Object-Oriented Programming*, M. Mezini, ed., vol. 6813 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, Berlin, Germany, 2011, pp. 79–104.

[21] T. Fritz, *Developer-Centric Models: Easing Access to Relevant Information*, doctoral dissertation, The Faculty of Graduate Studies, University Of British Columbia, Vancouver, Canada, Apr. 2011.

[22] F. Garcin, C. Dimitrakakis, and B. Faltings, "Personalized News Recommendation with Context Trees," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 105–112.

[23] X. Ge, D. Shepherd, K. Damevski, and E. Murphy-Hill, "How Developers Use Multi-Recommendation System in Local Code Search," *Proceedings: IEEE Symposium on Visual Languages and Human-Centric Computing*, Melbourne, Australia, July 2014, IEEE Computer Society.

[24] A. Grzywaczewski and R. Iqbal, "Task-specific information retrieval systems for software engineers," *Journal of Computer and System Sciences*, vol. 78, no. 4, 2012, pp. 1204–1218.

[25] R. Holmes, "Do developers search for source code examples using multiple facts?," *Proceedings: ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, Vancouver, Canada, May 2009, IEEE Computer Society, pp. 13–16.

[26] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona example recommendation tool," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, Sept. 2005, pp. 237–240.

[27] M. Howard, D. LeBlanc, and J. Viega, *19 Deadly Sins of Software Security*, 1st edition, McGraw-Hill/Osborne, Emeryville, CA, USA, 2005.

[28] W. Janjic and C. Atkinson, "Utilizing software reuse experience for automated test recommendation," *8th International Workshop on Automation of Software Test*, San Francisco, CA, USA, May 2013, IEEE Computer Society, pp. 100–106.

[29] M. Kaminskas, F. Ricci, and M. Schedl, "Location-aware music recommendation using auto-tagging and hybrid matching," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 17–24.

[30] S. Kpodjedo, F. Ricca, P. Galinier, and G. Antoniol, "Not all classes are created equal: toward a recommendation system for focusing testing," *Proceedings: 2008 International Workshop on Recommendation Systems for Software Engineering*, Atlanta, GA, USA, Nov. 2008, ACM, pp. 6–10.

[31] A. Kuhn, "On recommending meaningful names in source and UML," *Proceedings: 2nd International Workshop on Recommendation Systems for Software Engineering*, Cape Town, South Africa, May 2010, ACM, pp. 50–51.

[32] K. Kumar and S. Kumar, "A rule-based recommendation system for selection of software development life cycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, 2013, pp. 1–6.

[33] W. Lijie, F. Lu, W. Leye, L. Ge, X. Bing, and Y. Fuqing, "APIExample: An effective web search based usage example recommendation system for java APIs," *Proceedings: 26th IEEE/ACM International Conference on Automated Software Engineering*, Lawrence, KS, USA, Nov. 2011, IEEE Computer Society, pp. 592–595.

[34] A. Mahmoud and N. Niu, "Source Code Indexing for Automated Tracing," *Proceedings: 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, Honolulu, HI, USA, May 2011, ACM, pp. 3–9.

[35] Y. Malheiros, A. Moraes, C. Trindade, and S. Meira, "A Source Code Recommender System to Support Newcomers," *Proceedings: 36th IEEE Annual Computer Software and Applications Conference*, Izmir, Turkey, July 2012, IEEE Computer Society, pp. 19–24.

[36] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," *Proceedings: 6th IEEE International Working Conference on Mining Software Repositories*, Vancouver, Canada, May 2009, IEEE Computer Society, pp. 131–140.

[37] F. McCarey, M. . Cinnide, and N. Kushmerick, "Rascal: A recommender agent for agile reuse," *Artificial Intelligence Review*, vol. 24, no. 3–4, 2005, pp. 253–276.

[38] S. Muneer, M. Nadeem, and E. B. Allen, "Recommending Training Topics for Developers Based on Static Analysis of Security Vulnerabilities," *Proceedings: 52nd ACM Southeast Conference*, Kennesaw, GA, USA, Mar. 2014, ACM.

[39] N. Murakami and H. Masuhara, "Optimizing a search-based code recommendation system," *Proceedings: Third International Workshop on Recommendation Systems for Software Engineering*, Zurich, Switzerland, June 2012, IEEE Computer Society, pp. 68–72.

[40] E. Murphy-Hill, R. Jiresal, and G. C. Murphy, "Improving software developers' fluency by recommending development environment commands," *Proceedings: 20th International Symposium on the Foundations of Software Engineering*, Cary, NC, USA, Nov. 2012, ACM.

[41] M. Nadeem, E. B. Allen, and B. J. Williams, "Computer Security Training Recommender for Developers," *Poster Proceedings: 8th ACM Conference on Recommender Systems*, Silicon Valley, CA, USA, Oct. 2014, ACM.

[42] M. Nadeem, F. P. Haecker, B. R. Rice, B. N. Pilate, B. J. Williams, and E. B. Allen, "Evaluating Effectiveness of Approaches to Counter SQL Injection Attacks," *Poster Proceedings: IEEE SoutheastCon 2016*, Norfolk, VA, Mar. 2016, IEEE Computer Society.

[43] M. Nadeem, B. J. Williams, and E. B. Allen, "High false positive detection of security vulnerabilities: a case study," *Proceedings: 50th Annual ACM Southeast Regional Conference*, Tuscaloosa, AL, USA, Mar. 2012, ACM, pp. 359–360.

[44] F. Palma, H. Farzin, Y. Gueheneuc, and N. Moha, "Recommendation system for design patterns in software development: An DPR overview," *Proceedings: Third International Workshop on Recommendation Systems for Software Engineering*, Zurich, Switzerland, June 2012, IEEE Computer Society, pp. 1–5.

[45] Y. Pan, F. Cong, K. Chen, and Y. Yu, "Diffusion-aware personalized social update recommendation," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 69–76.

[46] B. Peischl, M. Nica, M. Zanker, and W. Schmid, "Recommending Effort Estimation Methods for Software Project Management," *Proceedings: IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, Milan, Italy, Sept. 2009, IEEE Computer Society, vol. 3, pp. 77–80.

[47] M. S. Pera and Y.-K. Ng, "What to read next? Making personalized book recommendations for K-12 users," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 113–120.

[48] L. Ponzanelli, "Holistic Recommender Systems for Software Engineering," *Proceedings: 36th International Conference on Software Engineering*, Hyderabad, India, June 2014, ACM, ICSE Companion 2014, pp. 686–689.

[49] M. F. Porter, *Readings in information retrieval*, chapter An algorithm for suffix stripping, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010, pp. 313–316.

[50] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds., *Recommender Systems Handbook*, Springer, New York, NY, USA, 2011.

[51] S. Richardson, *Dynamically Recommending Design Patterns*, doctoral dissertation, Department of Mathematics and Computer Science, College of Arts and Science, Stetson University, DeLand, FL, USA, 2011.

[52] M. P. Robillard and Y. B. Chhetri, "Recommending reference API documentation," *Empirical Software Engineering*, July 2014.

[53] K. Roher and D. Richardson, "A proposed recommender system for eliciting software sustainability requirements," *Proceedings: 2nd International Workshop on User Evaluations for Software Engineering Researchers*, San Francisco, CA, USA, May 2013, IEEE Computer Society, pp. 16–19.

[54] R. Shimada, Y. Hayase, M. Ichii, M. Matsushita, and K. Inoue, "A-SCORE: Automatic software component recommendation using coding context," *Proceedings: 31st International Conference on Software Engineering*, Vancouver, Canada, May 2009, ACM, pp. 439–440.

[55] I. Steinmacher, I. S. Wiese, and M. A. Gerosa, "Recommending Mentors to Software Project Newcomers," *Proceedings: Third International Workshop on Recommendation Systems for Software Engineering*, Zurich, Switzerland, June 2012, IEEE Computer Society, pp. 63–67.

[56] D. Surian, N. Liu, D. Lo, H. Tong, E.-P. Lim, and C. Faloutsos, "Recommending People in Developers' Collaboration Network," *Proceedings: 18th Working Conference on Reverse Engineering*, Limerick, Ireland, Oct. 2011, IEEE Computer Society, pp. 379–388.

[57] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 1st edition, Addison-Wesley, Boston, MA, USA, 2005.

[58] R. Terra, M. T. Valente, R. S. Bigonha, and K. Czarnecki, "DCLfix: A recommendation system for repairing architectural violations," *Proceedings: 3rd Brazilian Conference on Software: Theory and Practice*, Natal, Brazil, Sept. 2012, pp. 1–6.

[59] K. Toutanova and C. D. Manning, "Enriching the Knowledge Sources Used in a Maximum Entropy Part–of–Speech Tagger," *Proceedings: Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, Hong Kong, Oct. 2000, ACM, pp. 63–70.

[60] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth, "Learning from project history: A case study for software development," *Proceedings: 2004 ACM Conference on Computer Supported Cooperative Work*, Chicago, IL, USA, Nov. 2004, ACM, pp. 82–91.

[61] P. Viriyakattiyaporn and G. C. Murphy, "Challenges in the user interface design of an IDE tool recommender," *Proceedings: ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, Vancouver, Canada, May 2009, IEEE Computer Society, pp. 104–107.

[62] Z. Xiao-lin, C. Chi-Hung, D. Chen, and R. K. Wong, "Non-functional Requirement Analysis and Recommendation for Software Services," *Proceedings: 20th IEEE International Conference on Web Services*, Santa Clara, CA, USA, June 2013, IEEE Computer Society, pp. 555–562.

[63] Y. Zhang and M. Pennacchiotti, "Recommending branded products from social media," *Proceedings: 7th ACM conference on Recommender systems*, Hong Kong, Oct. 2013, ACM, pp. 77–84.

[64] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the Value of Static Analysis for Fault Detection in Software," *IEEE Transactions on Software Engineering*, vol. 32, no. 4, Apr. 2006, pp. 240–253.

APPENDIX A

DETAILS OF SYSTEMATIC MAPPING STUDY OF RECOMMENDER SYSTEMS

A.1 Search Strings Used

We used the following search string

```
Recommender system for software engineer

Recommender system for software developer

Recommendation system for software engineer

Recommendation system for software developer
```

The advanced search string is given as under.

```
(("recommender system") OR ("recommendation system"))

AND

(("software developer") OR ("software engineer"))
```

A.2 The Databases Used for Literature Search

The following research databases were used for conducting the literature search.

- ACM Digital Library
- Google Scholar
- IEEEXplore
- ScienceDirect
- Scopus

The search initially returend 927 articles, out of which 141 duplicate studies were excluded; hence, bringing the remaining number of studies to 786. After carefully going through the title of the article, we excluded 653 more articles; hence, bringing the reamining number of articles to 133. Then, we excluded 72 more articles after reading the abstract;

hence, bringing the total down to 61. Finally, after reading conclusions and results, 7 more articles were excluded; hence, bringing the total to 54 articles.

APPENDIX B

HUMAN SUBJECT EXPERIMENT DETAILS

B.1 Human Subject Evaluation

The template of form used for evaluating the different training articles is shown in Figure B.1.

The template of form used for ranking the different training articles is shown in Figure B.2.

Evaluation form

Description:

Based on the vulnerability type *"Cross Site Scripting"* detected in the code, the attached article has been recommended as a training resource to understand and fix the vulnerable code.

Task:

Being a software security expert, please evaluate the attached article `VT11` using form below.

Evaluation:

Please select the most appropriate options for each question below.

1) Relevance of information to flagged vulnerability

① Not relevant ② ③ ④ ⑤ Highly relevant

Any additional comments:

2) Amount of information in the article for training purpose

① Too little ② ③ Just right ④ ⑤ Too much

Any additional comments:

3) Suitability of information as training material

① Not suitable ② ③ ④ ⑤ Highly suitable

Any additional comments:

Figure B.1

Form used for evaluation of training articles

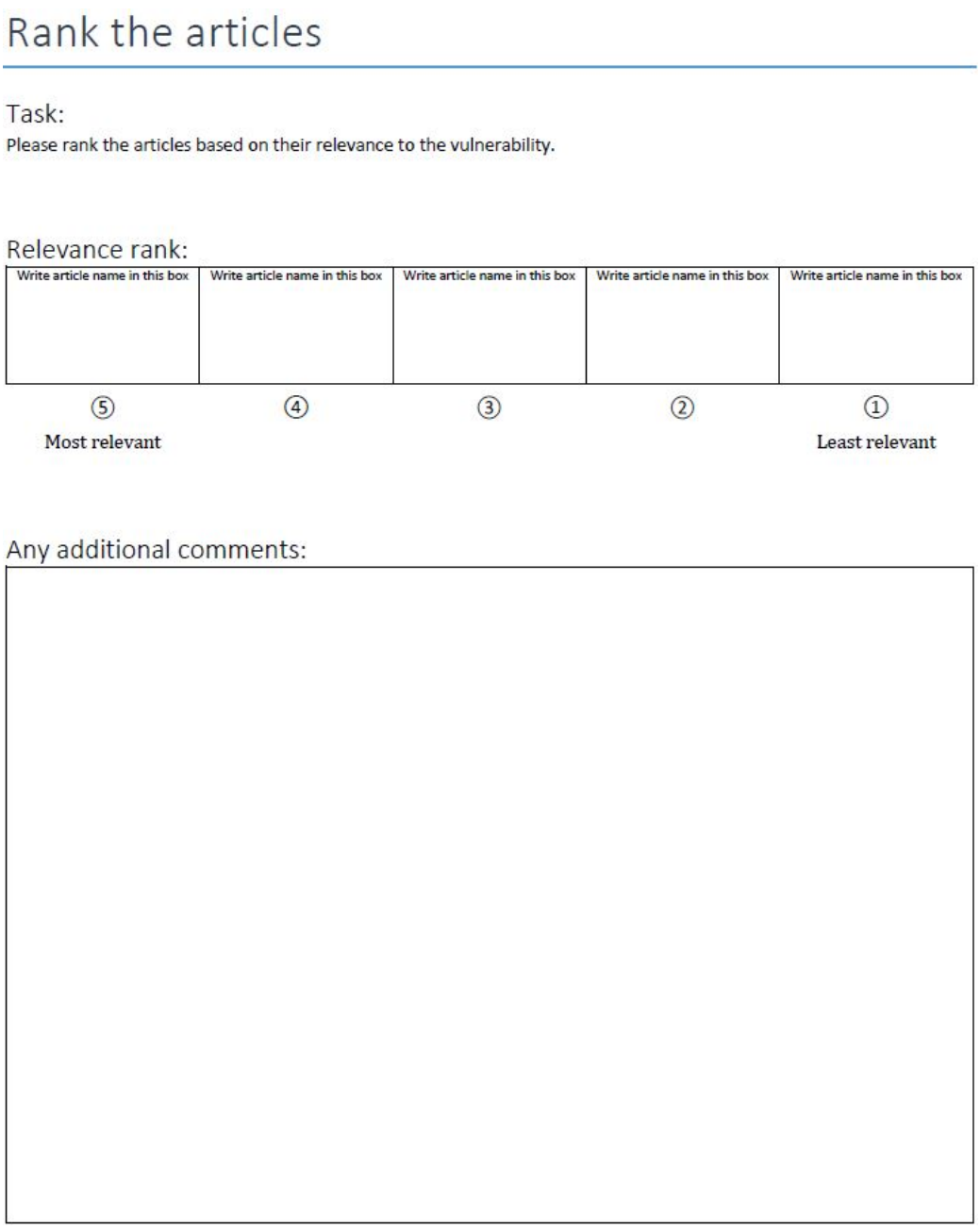


Figure B.2

Form used for ranking of training articles