

# University of Cincinnati

Date: 11/7/2018

**I, Kavya Reddy Mahakala, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.**

It is entitled:

**Identifying Security Requirements using Meta-Data and Dependency Heuristics**

Student's name: **Kavya Reddy Mahakala**

This work and its defense approved by:

Committee chair: Nan Niu, Ph.D.

Committee member: Raj Bhatnagar, Ph.D.

Committee member: Carla Purdy, Ph.D.



31795

**IDENTIFYING SECURITY REQUIREMENTS USING META-DATA AND  
DEPENDENCY HEURISTICS**

A thesis submitted to the  
Graduate school  
of the University of Cincinnati.  
In partial fulfillment of the  
Requirements for the degree of  
Master of Science

In the Department of Engineering and Applied Sciences

By

Kavya Reddy Mahakala

University of Cincinnati

Fall 2018

Committee Chair: Dr. Nan Niu, Ph.D.



## **ABSTRACT**

There are several security requirements identification methods proposed by researchers in up-front requirements engineering (RE). However, in agile and open source software projects, developers generally tend to discuss the requirements through comments, handle the implementation through external resources, and use available libraries for development. Considering the above factors, in this work we have tried to find and analyze heuristics which would help to classify the requirements into security or non-security requirements. Through this work, we aim to identify as many security requirements as we can by using the proposed heuristics set. We apply the approach to a web application in practice which uses open source libraries for development to understand and analyze our heuristics in depth.

## **ACKNOWLEDGMENT**

Apart from my hard work, I truly believe that this work would not have been possible without the constant and continuous support from Dr. Nan Niu. His ideas and directions have helped me in discovering different dimensions around this work. I would like to convey my deepest gratitude for the help and guidance he has always provided. I would to take this opportunity to thank my committee members – Dr. Raj Bhatnagar and Dr. Carla Purdy for their valuable time. I sincerely appreciate the constant help, support and knowledge provided by Wentao Wang.

## TABLE OF CONTENTS

Title	1
Abstract	3
Acknowledgment	4
Table of Contents	5
Index	6
List of Figures	8
List of Tables	9

## INDEX

<b>1. INTRODUCTION</b>	<b>10</b>
1.1. Web Engineering	13
<b>2. BACKGROUND AND RELATED WORK</b>	<b>16</b>
<b>3. CLASSIFICATION USING META-DATA HEURISTICS</b>	<b>21</b>
3.1. Datasets and Meta-Data Heuristics Selection	21
3.1.1. Datasets and Answer Sets	21
3.1.2. Meta-Data Heuristics Selection	25
3.2. Linear Classifier Based Approach	31
3.2.1. Logistic Regression Model	33
3.2.2. Linear Classifier	33
3.3. Measurement and Results Analysis	35
3.3.1. Measurement	35
3.3.2. Results Analysis	36
3.3.3. Discussion and Improvements	39
<b>4. INTRODUCTION TO SEMANTIC HEURISTICS</b>	<b>42</b>
4.1. Security Related Semantic Aspects	42
4.1.1. Conditional Trigger	43
4.1.2. Requirements Centric Implications	44
4.1.3. Depreciated Dependency	46
4.1.4. Derived Implications	47
4.2. Working of Packages	48

4.2.1. Package Dependency	49
4.2.1.1. Package Dependency Graphs	55
4.3. Potential Heuristics	58
<b>5. CASE STUDY</b>	61
5.1. Introduction	61
5.1.1. Background	61
5.1.2. Requirements	63
5.1.3. User Interfaces	65
5.1.4. Validators	68
5.2. Results and Analysis	68
<b>6. CONCLUSION</b>	80
6.1. Limitations	82
6.2. Future Work	83
<b>REFERENCES</b>	84



## LIST OF FIGURES:

Figure 1.1. Web Application Architecture	13
Figure 1.2. MVC Modelling	14
Figure 3.1. Requirements in Issue Tracking System	22
Figure 3.2. Security Related URL Examples	29
Figure 3.3. Linear Classifier-Based Approach Architecture	32
Figure 3.4. Linear Classifier-Based Security Requirement Identification Approach	37
Performance Under Different Threshold	
Figure 4.1. Illustration for Conditional Dependency	43
Figure 4.2. Illustration of Requirement Centric Implications	45
Figure 4.3. Illustration for Depreciated Dependency	46
Figure 4.4. npm Audit Reports Containing Severity of Security Vulnerabilities	54
Figure 4.5. Package Dependency Graph	56
Figure 4.6. npm Package List Using npm ls/ npm list Command	57
Figure 5.1. Sample Packages Installed for the Web Application.	66
Figure 5.2. Login Page	66
Figure 5.3. Interface of the Web Application	67
Figure 5.4. Flow of Attack	72
Figure 5.5. Code Snippet of Attack Introduced in Package	73
Figure 5.6. Code Snippet of Call made by Web Application	74
Figure 5.7. View of the System After the Attack	75
Figure 5.8 View on the Attacker's Side.	75

## LIST OF TABLES

Table 3.1. Basic Information of Projects	22
Table 3.2. Sample of Keywords from Projects	24
Table 3.3. Statistical Analysis of Complexity Metrics	28
Table 3.4. Comparison between NFR-C, RMs, and LC	38
Table 4.1. Overview of npm Package Manager	53
Table 5.1. Subset of Requirements that were Framed for our Emailing System Web Application	63
Table 5.2. Classification and Analysis using Meta-Data Heuristics	70
Table 5.3. Classification and Analysis using Semantic Heuristics	78
Table 5.4. Comparison between Meta-Data Heuristics-Based Labels and Semantic Heuristics-Based Labels	79

## CHAPTER 1: INTRODUCTION

Security refers to a class of non-functional requirements (NFRs) related to system confidentiality, integrity, and availability [1]. It is a fundamental requirement and research direction for the Internet of Things (IoT) [2, 3], and also a key component of Industry 4.0 (I4.0) [4, 5].

Experience indicates that considering security early in the software life cycle can help address security problems, such as reducing the defect density ratio (e.g., number of bugs per thousand lines of code) [6]. Early detection of security requirements enables engineers to incorporate security into the initial architectural design [7]. Security requirements may be implicit, hidden, and spread out over different parts of mostly textual requirements specifications. Identifying those security requirements requires expertise and experience. Unfortunately, security experts are not always available. At the same time, manually identifying security requirements is a tedious and error-prone task. Neglecting security requirements early in a project can cause substantial security problems later.

Observations on open-source software (OSS) development indicate that stakeholders rarely classify whether a requirement is security related while it is reported. In most cases, they identify security requirements through discussion during the implementation cycle. However, experiments also show that failing to recognize security requirements will lead to buggy implementation and re-opening issues.

Therefore, having an automated or a semi-automated tool support for identifying security requirements in OSS projects is the key to successfully implementing those projects.

A considerable number of studies have been done on automated detection of security requirements [8, 9]. However, they are labor intensive. Cleland-Huang et al. [7] proposed NFR-classifier, an automated approach based on information retrieval methods for detecting and classifying NFRs. Mahmoud and Williams [10] proposed another automated approach which exploits the textual semantics of software functional requirements (FRs) to infer potential quality constraints enforced in the system. Their research demonstrated that those methods can achieve high accuracy in traditional or up-front RE. As a precursor to our work, we applied these methods to three open source software (OSS) projects which will be described later, and the results show that none of them achieved similar performance as in up-front RE projects. Up-front RE is known to specify requirements formally and then baseline in terms of development [47]. Furthermore, the goal of those research approaches is providing general methods for all NFRs, not specific for security requirements. Therefore, security-specific semantic information, such as CVE (Common Vulnerabilities and Exposures)<sup>1</sup> ID or corrupt packages, is not included in their approaches.

Motivated by the above observations, we propose in this work a novel and efficient approach for identifying security requirements in OSS development. In this work, we make the following contributions: 1) building models with metrics related to requirements complexity and resources external to the software project; and 2) finding an optimal way

---

<sup>1</sup> <https://cve.mitre.org>

to integrate all models with NFR-classifier 3) backtrack through the results to check the heuristics that could be applied in order to make the results better 4) illustrating how the heuristics could actually contribute towards correcting the false negatives using a case study. Our initial results consisting the meta-data heuristics show that the approach can achieve on average 92.31% recall and 62.94% precision in three OSS projects, which significantly outperforms prior approaches. Although, these results are in itself a good contribution towards automating the detection of security requirements in web related projects. We further try to check ways of dealing with the false negatives through our semantically enriched heuristics.

In real world web projects, each project is unique and has its own implementation techniques. We often notice the usage of many Open Source Resources like libraries and packages. Therefore, building a completely automated model which is common to all such web projects can be extremely difficult. There can be projects where a set of defined metrics may not be applicable to a project. As a result, the supervision of a software engineer would be required to some extent in order to judge, correct, and improve the consideration of factors in detecting any security vulnerabilities in a requirement. In this work, we not only consider the extent to which the process of detecting security requirements can be automated, but also take into account the factors which a software engineer or any human expert can consider in making the security aspect related to any requirement drilled down to both descriptive and implementation levels.

This work contains a case study which is a web-based project used to describe how different web projects may need different factors for automation, but might hold on a

common ground in certain areas that the requirements experts and security experts should not ignore.

## 1.1. Web Engineering

As this work discusses much about web applications and the working of it, let us understand how a web application is built and engineered. The way in which web applications are being built has been drastically evolving. This evolving nature includes the frameworks involved, the implementation of services, and the languages considered while building web applications.

In modern day, we can model the web architecture as shown in Figure 1.1.

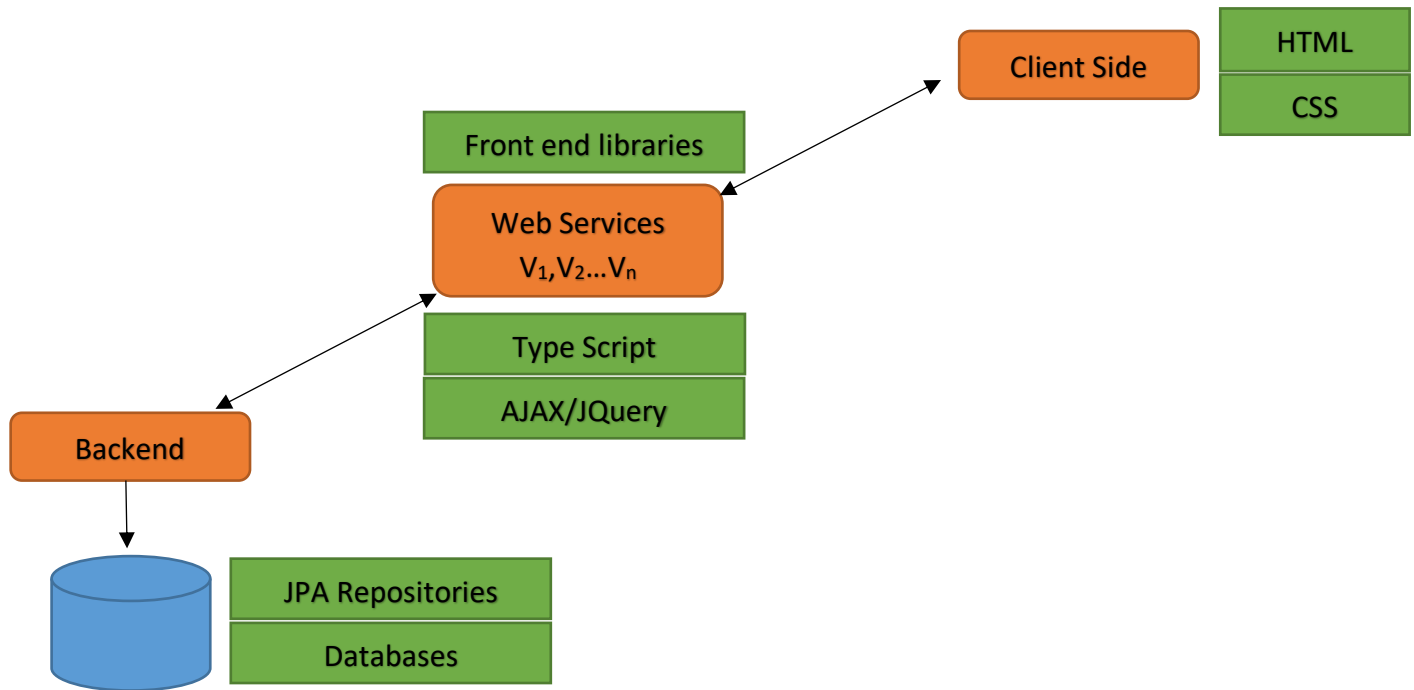


Figure 1.1: Web Application Architecture

The web architecture shown in Figure 1.1 mainly has three major components- Web services, Backend, and Client side. The communications between backend, client side with

web services are bi-directional. In the initial phases, the web architecture consisted of separate web services acting independently. But, with the advent of the modern implementation methodology the web services have the capability of different services working together.

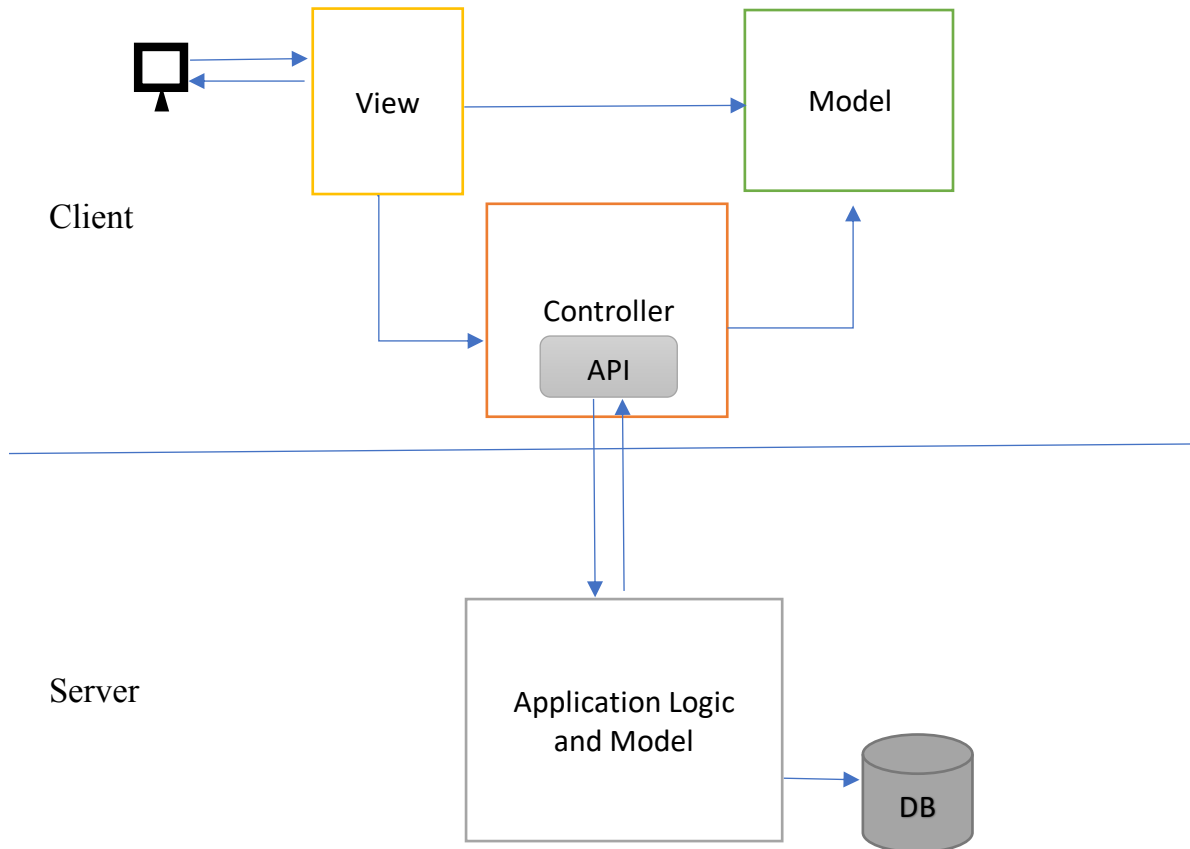


Figure 1.2: MVC Modelling

We generally encounter the concept of MVC in the implementation of web applications. MVC is intended to bring model, view and controller together. In a web application we often communicate information between client and server through web API's. This can be modelled as shown in Figure 1.2.

An important aspect of web applications is the usage of packages which enable reuse of code and save time. Further in this work, we learn more about the effects of packages in terms of security.

In this work we use requirements from web applications to check for security implications. This work deals with the meta-data and semantic heuristics that are applied on the requirements in order to study how they affect the classification of security requirements and non-security requirements.

The remainder of this thesis is organized as follows. Chapter 2 reviews related work in security requirements identification. In Chapter 3, we describe the rationale of datasets and metrics selection. In this chapter, we consider the metrics selected (meta-data heuristics) and introduce a linear classifier-based approach. In this chapter, we further assess the performance of our approach. In Chapter 4, we describe more semantically enriched heuristics which have the potential to solidify our goal of security requirements classification. In Chapter 5, we introduced a case study of a web-based project and test our initial linear-classifier based approach with the case study. We further analyze the semantically enriched heuristics on the case study and build a relation between the results provided by classifier when using the meta-data heuristics and the results produced on using the semantically enriched heuristics. Chapter 6 concludes the thesis and discusses prospects for future work and limitations.



## CHAPTER 2: BACKGROUND AND RELATED WORK

Existing research effort applied different machine learning algorithms to identify and classify NFRs. In this chapter, we introduce previous proposed approaches based on different machine learning algorithms and their performances on security requirements identification.

Cleland-Huang et al. [7] described NFR classifier, an automated method based on the assumption that certain keywords can be used to distinguish different types of NFRs. A set of predefined requirements was used to derive different sets of weighted keywords called indicator terms for different NFRs. Indicator terms were then used as queries to retrieve NFRs from various software artifacts. An industrial case study was conducted to evaluate the proposed approach. The result showed that NFR classifier can recall 82.8% security requirements. At the same time, the result also showed a very high ratio of false positive (7.1% precision). Their proposed approach benefited from the textual features of requirements, and ignored other requirements features, e.g., stakeholder's information and lexical information. We conjecture that contextual and lexical features are complementary with the textual feature on identifying security requirements and thus classifiers built with contextual information may be possible to integrate with NFR classifier and improve the performance of the security requirements identification approach.

Unlike NFR classifier which derives indicator terms from a pre-labeled set of requirements, Mahmoud and Williams [10] proposed a method which discovers and assesses the query terms by using un-labeled datasets. Their proposed approach first grouped terms of requirements into different clusters based on their semantic similarity.

Then average pairwise similarity between clusters and NFR categories (e.g., accessibility, interoperability, and security) was calculated. A cluster was only assigned to the NFR category with highest average pairwise similarity. Requirements then were automatically classified into different NFR categories based on their semantic similarity to the clusters of terms representing individual NFRs. This approach adopts crisp clustering to ensure clear cut results, i.e. one term can belong to one NFR only. However, the same term can indicate different things. For instance, “denial-of-service” is related to both security and availability. Thus, crisp clustering may cause occurrence of incomplete set of representative terms. Furthermore, it may harm the performance of NFR identification.

Kurtanović and Maalej [11] applied Support Vector Machine (SVM) with 11 different lexical features, such as text length, fractions of nouns and verbs, and sentence syntax tree height to build two classifiers. The binary classifier was used to distinguish functional requirements (FRs) and NFRs. The multi-class NFRs classifier worked on the result of the binary classifier to identify usability, security, operational, and performance NFRs. The proposed approach was applied to two datasets, and results showed that this method could find 90.1% security requirements and filter out most non-security requirements (precision 90.0%).

However, unlike datasets used in this research, requirements specifications in OSS projects tend to be organized by functionality, with NFR scattered widely across multiple documents [7]. Therefore, there is no clear boundary between FRs and NFRs. Moreover, the requirements stored in issue tracking systems are unstructured and seldom obey grammar and punctuation rules [12]. Thus, this approach is not suitable for OSS projects.

Munaiah et al. [13] proposed an unsupervised approach for detecting security requirements. They trained their One-Class Support Vector Machine with the Common Weakness Enumeration (CWE)<sup>2</sup>, a formal list of software weakness types intended to serve as the common language for describing software security weaknesses in architecture, design, or code. The authors' assumption is that the language used to describe security requirements and that used to describe weaknesses overlap thus it could be a good candidate to train the classifier. As Gibiec et al. [14] pointed out, term mismatch problem commonly occurs in software artifacts. Additionally, specific terms contained in requirements which indicate concerns like security and dependability are domain/application dependent [18]. Therefore, the performance of this approach is lower than other approaches that we discussed earlier (i.e., 70.48% recall and precision level is 67.35%).

To the best of our knowledge, there is no approach using features related to requirements complexity and external resources to identify security requirements. However, those features are widely applied in other research domains, such as software quality prediction [15], software complexity evaluation [16], and vulnerability detection [17]. In our work, we take advantage of these features to train a binary security requirements classifier.

There has been much research which supports how security vulnerabilities are different when compared to bugs. Camillo et al. [33] have conducted an empirical study on

---

<sup>2</sup> <https://cwe.mitre.org>

Chromium project to emphasize the difference between security vulnerabilities and other types of bugs in software. npm is a package manager and contains security vulnerabilities which have come into light due to the large number of security reports that have been received. Cox et al. [34] have explained through their study about how updated packages are less prone to security vulnerabilities when compared to outdated packages installed. A similar analysis was done by Massacci et al. [35] who studied the evolution of Firefox. In their study, they emphasize that old versions that are not maintained have a greater number of security vulnerabilities which are carried into newer versions when the users slowly upgrade.

As discussed earlier, packages have security vulnerabilities which get carried to their dependencies as well. Lauinger et al. [37] demonstrated this point by analyzing security vulnerabilities on the use of JavaScript libraries in client-side programming. Decan et al. [36, 42], have compared npm with other package managers to study the topology of package dependency networks. There has been much research related to package dependencies lately. Decan et al. performed a study [40] to explain the number of packages affected by security vulnerabilities in npm package manager, the duration of time during which the packages stay vulnerable, the discovery of security vulnerabilities, and the fix of those vulnerabilities. In [40], Decan et al. conducted an empirical study of security reports based on their severity and studied how security vulnerabilities propagate through dependent packages.

In [31], Wittern et al. conducted a study about npm packages and reported their versioning, dependencies, frequency of updating the packages by package maintainers and

policies. This work discusses how the package meta-data in package.json file function. The work gives description about the evolution of categorization in npm packages. This work deals with studying the evolution of ecosystem. Similarly, in [32] Mitropoulos et al. studied the software ecosystem and used FindBugs, a software tool to identify security bugs in open source libraries within the ecosystem.

## **CHAPTER 3: CLASSIFICATION USING META-DATA HEURISTICS**

In this Chapter, we discuss about the Meta-Data Heuristics that are involved in an Open Source Software project, the factors why we choose particular meta-data heuristics as features to pass to the linear classifier built in our study, which aims at classifying the requirements into security or non-security requirements.

### **3.1. Datasets and Meta-Data Heuristics Selection**

The first challenge in building security requirements identification classifier is to find candidate meta-data heuristics. After manually analyzing three web-based OSS projects, we found that two types of meta-data heuristics are suitable to distinguish security requirements from non-security requirements. In this section, we first introduce the three datasets we used to evaluate our approach and then discuss meta-data heuristics selection.

#### **3.1.1. Datasets and Answer Sets**

In our study, we analyzed the requirements of three OSS projects: Apache Axis2/Java (Axis2), Drools, and GeoServer. We selected these projects as the subject systems due to three reasons. First, all of them are successful and long-lived projects. Second, all resources including requirements and source code are available. At last, all three projects are web-based systems and security is one of the core aspects of these projects, therefore, identifying and realizing security requirements are important tasks for developers of these three projects. These projects come from different application domains, and all of them are written in Java. Axis2 is a web services engine funded by Apache Software Foundation since August 2004. The newest version of Axis2 (1.7.4) was released in December 2016. Drools is a business rule management system developed by Red Hat.

The stable release of Drools that we studied is 6.5.0, which was published in December 2016. GeoServer is a geographic system which allows users to edit and share geospatial information. The stable version of GeoServer that we studied is 2.11.0 which was published in March 2017.

Table 3.1: Basic Information of Projects

Project	Domain	# of req.s	# of security req.s
Axis2	Web server	5751	109
Drools	Business	326	33
GeoServer	Geo information	8172	651

The screenshot displays the JIRA issue tracking interface for Axis2. The issue title is "Allow for sharing XSD schemas between services" (AXIS2-3354). The issue is categorized as an "Improvement" with a "Major" priority and is currently in an "OPEN" status. The description states: "Suppose I have the following structure: The WSDLs for MyService1 and MyService2 are in the following folders, respectively: contextpath/WEB-INF/services/MyService1/META-INF/". An attachment named "patch.txt" (1 kB) is listed. The "Activity" section shows a comment from Amila Chinthaka Suriarachchi dated 26/Dec/10 10:15, discussing the solution proposed by Deepal and the removal of relative file locations for security reasons. The interface includes sections for "Details", "Description", "Attachments", "People" (Assignee: Srinath Perera, Reporter: Mauro Molinari), "Dates" (Created: 20/Nov/07 16:17, Updated: 04/Jan/11 08:59), and "Activity".

Figure 3.1. Requirement in Issue Tracking System

Table 3.1 shows some basic information about these three projects. Figure 3.1 shows a security requirement: AXIS2-3354. The reporter depicted this new requirement in the description part (Figure 3.1 A) when she created it. She also provided an attachment to further explain the requirement (Figure 3.1 B). Then, developers communicated and refined this requirement by providing comments (Figure 3.1 C). Figure 3.1 D shows all stakeholders of this requirement.

A two-step analysis is applied to generate answer sets of security requirements. In the first steps, we use pre-defined security indicator terms to retrieve candidate security requirements from three data sets. Then, in the second step, two experts classify the candidate links as security or non-security requirements. Cleland-Huang et al. [7] conducted an experiment to test the performance of a security requirements retrieval method: keywords searching. The results show that keywords searching can achieve high recall (i.e., over 70%) but low precision (i.e., less than 35%). On the other hand, manually going through all requirements, especially big datasets such as Axis2 and GeoServer, costs much of experts' working time and therefore may have the negative impact on experts' judgment (i.e., experts make more incorrect decisions). Therefore, we believe retrieving candidates and present them to experts will attract their attention, thus may increase the quality of answer sets.



Table 3.2: Sample of Keywords extracted from projects

Project	Project dependent indicator terms
Axis2	Apache rampart, credentials, MEP (message exchange pattern), stack overflow exception, socket, SOAP (simple object access protocol), threadsafe, WS-Security, WS-Addressing
Drools	ACL (access control list), security-module-admin.properties, security-policy.properties, permission
GeoServer	GeoServerSecurityFilterChain, GeoServerSecurityFilterChainProxy, KeyStoreProvider, OAuth2, ResourceAccessManager

Another purpose of the first step is to retrieve as many security requirements as possible. Therefore, we not only use security indicator terms defined by Cleland-Huang et al. [7], but also extract more indicator terms from another classic literature [1] which systematically describes concepts and taxonomy about security. In addition, we also find that, developers of three datasets describe their security concerns and strategies in separate projects documents. For instance, the Drools team explains concepts related to security management and technologies they adapt to help them implement them in the Drools documentation [19]. Thus, we extract keywords from those documents and include them to our indicator terms. Table 3.2 represents all keywords extracted from projects' documents. We also asked our experts to go through all the remaining requirements (i.e., requirements are not retrieved by keywords searching) in Drools. Experts did not find any additional security requirements. This small testing increases our confidence that our answer sets include all the security requirements.

The main purpose of the second step is including only security requirements in three answer sets. Two experts analyze all candidates and find security requirements independently. Requirements which are classified as security requirements by both experts will be included in our answer sets immediately. According to Cohen's Kappa analysis, two analysts achieved almost perfect agreement [20] (i.e. 0.89, 0.73, and 0.97 on Axis2, Drools, and GeoServer respectively). For requirements that two experts have different opinions, they make the final decision through a face-to-face discussion. The remaining requirements, including requirements which are not retrieved in the first step and requirements which are not considered as security requirements in the second step, will automatically be considered as non-security requirements.

### **3.1.2. Meta-Data Heuristics Selection:**

In this section, we classify our meta-data heuristics into two categories: complexity and external resources. For each of these heuristics, we not only provide the hypotheses for the discriminative power and the predictability, but also provide the rationale for why these metrics can work for security requirements identification.

#### ***Complexity:***

Security experts claim that complexity is the enemy of security [6, 21]. Complexity in requirements can lead to complex code which contains subtle vulnerabilities that are difficult to test and diagnose, providing more chances for attackers to exploit [18]. For this reason, we set up the following hypothesis on complexity in requirements.

#### *H Comments Complexity:*

Security requirements in OSS projects involve more discussions than non-security requirements. A requirement containing more comments has a higher chance of having an interaction with other requirements, leading to more code having input from the external source to modules implemented by other developers without properly understanding the security concerns or assumptions of the modules [17]. We first tested four meta-data heuristics, including number of comments/requirement (#C), number of nouns in comments/requirement (# N), number of verbs in comments/requirement (# V), and average length of comments/requirements (ALC), by comparing their distribution difference between security and non-security requirements. For each dataset, we chose subsets of security and non-security requirements via simple random sampling (SRS) [22]. Shapiro-Wilk test accepts distribution normality for all the above meta-data heuristics in both security and non-security requirements (i.e., p-values greater than 0.05). Therefore we applied student t-test to test statistic difference between two sets of the meta-data heuristics. Table 3.3 shows the testing results. Only ALC has the significant difference between security and non-security requirements. Therefore, only ALC is used to measure comments complexity.

#### *H Stakeholders Complexity:*

Security requirements in OSS projects involve more stakeholders than non-security requirements.

We used the number of stakeholders/requirement (# S) to measure the stakeholders' complexity. Issue tracking system used by three projects (i.e., JIRA) predefined several

stakeholders: creators, assignees, comment authors, watchers, and voters. Creators, assignees, and comment authors provide their opinions and refinement of requirements via commenting on requirements. Different people have different opinions and preferences. Therefore, the more stakeholders provide their opinion, the more complicated the requirements are. In addition, human errors are the most frequent type of errors in the requirements [23]. The errors also indicate the high possibility of vulnerability. For watchers and voters, JIRA only shows how many of them, not who they are. That means they may or may not overlap with creators, assignees, and comment authors. If they do not belong to previous three types of stakeholders, that means they do not provide any opinions or refinements. Therefore, we ignore the watchers and voters when we calculate stakeholder's complexity. The student t-test results (Table 3.3) show that there is a significant difference between security and non-security requirements.

#### ***External Resources:***

Stakeholders tend to provide external resources, e.g., URLs to documents on other websites, to provide the rationale for their refinement and explain their solutions. After analyzing three OSS projects, we found that three external resources have potential to help to distinguish security and non-security requirements. From our observations, the stakeholders add URLs to the requirements while they want to use information provided by the URLs, e.g., solutions for the current requirements, project dependability's, and detailed description of requirements. However, not every URL can indicate the security requirements. After analyzing three OSS projects, we found that only two types of URLs indicate the security requirements.

Table 3.3. Statistic Analysis for Complexity Metrics: in the third and fourth column, we show mean  $\pm$  standard deviation, and Shaprio-Wilk test results are showed in between brackets. in the fifth column \* indicates significant difference.

Project	Metric	Sec. Req.s	Non-Sec. Req.s	Significance
Axis 2	#C	4.24 $\pm$ 4.67 (.08)	3.97 $\pm$ 4.93 (.08)	
	#N	11.23 $\pm$ 8.23 (.09)	12.65 $\pm$ 9.72 (.08)	
	#V	7.52 $\pm$ 6.26 (.11)	6.71 $\pm$ 7.64 (.09)	
	ALC	33.16 $\pm$ 21.37 (.1)	15.26 $\pm$ 22.69 (.07)	*
	#S	7.26 $\pm$ 3.21 (.08)	4.31 $\pm$ 4.59 (.12)	*
Drools	#C	6.23 $\pm$ 3.76 (.09)	5.44 $\pm$ 5.24 (.07)	
	#N	10.23 $\pm$ 11.73 (.06)	9.72 $\pm$ 9.27 (.09)	
	#V	7.43 $\pm$ 7.53 (.08)	6.99 $\pm$ 6.17 (.11)	
	ALC	29.73 $\pm$ 32.61 (.07)	17.29 $\pm$ 29.96 (.09)	*
	#S	6.33 $\pm$ 5.31 (.1)	4.11 $\pm$ 4.23 (.12)	*
GeoServer	#C	5.49 $\pm$ 7.62 (.09)	4.24 $\pm$ 6.52 (.11)	
	#N	11.33 $\pm$ 7.54 (.13)	10.58 $\pm$ 3.72 (.08)	
	#V	5.43 $\pm$ 5.29 (.09)	6.34 $\pm$ 4.23 (.08)	
	ALC	24.52 $\pm$ 28.31 (.07)	16.33 $\pm$ 23.42 (.14)	*
	#S	8.21 $\pm$ 4.27 (.09)	3.62 $\pm$ 3.69 (.1)	*

In the Type I example, the URL points to the document which is mainly about security-related requirements (Figure 3.2(a)). In this case, the last part of the URL, which is the name of the security document (e.g. “selfadmin.html”), contains security indicator terms. In the Type II URL (Figure 3.2(b)) points to a document which contains both security and non-security requirements. However, it uses “#” to refers to the title of the specific section/chapter in this document. If the title of this specific section/chapter (e.g. “Character Encodings”) contains security indicator terms, we classify it as a security related-URL.

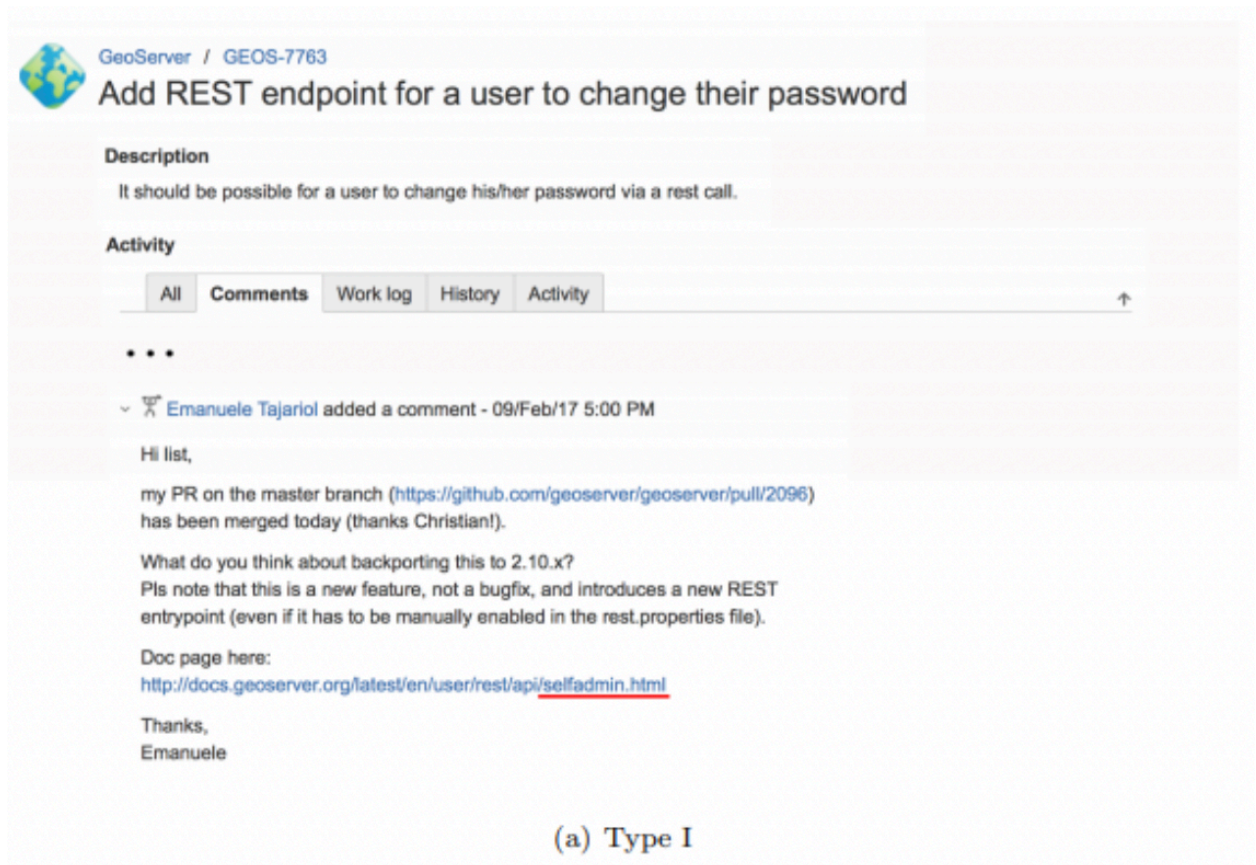


Figure 3.2 (a) : Security Related URL Example

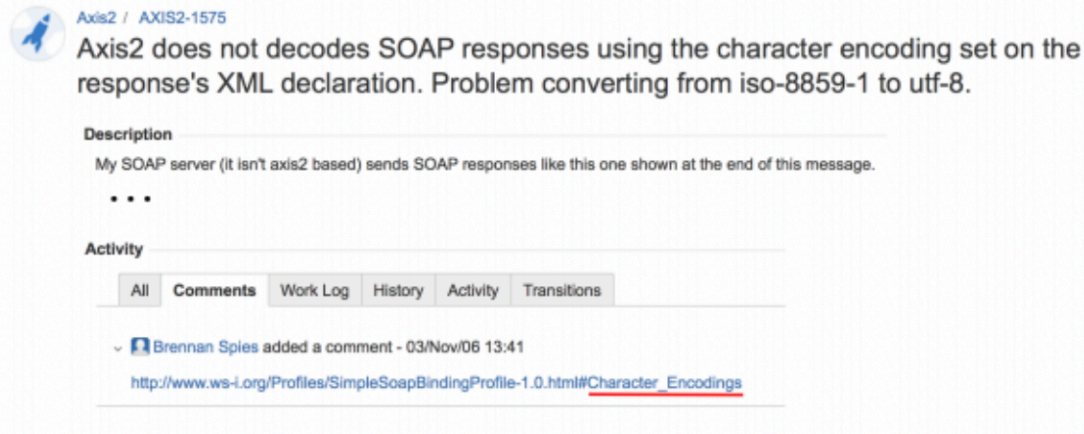


Figure 3.2 (b): Security Related URL Example

Therefore, we provide following hypothesis:

*H<sub>Security URLs</sub>:*

Security requirements in OSS projects involve more security related URLs than non-security requirements.

Another external resource provided by stakeholders is the commits from version control system, e.g. CVS, GitHub, etc. These systems record the diff for each commit. Security experts claim that the permission system, an approach to restricting system access to authorized users, is not sufficient to prevent sensitive data from being leaked [24]. Therefore, any outside sources can contaminate the sensitive data [25]. Thus, we are only interested in the commits which contain changes related to data exchange. From our observations, there are two types of code change strongly related to data exchange: 1) the change contains “=”; and 2) the change contains “get” or “set” methods. Thus, only the commits which contain data exchange related changes are counted as security-related commits.

We provide the following hypothesis for security-related commits:

*H Security Commits:*

Security requirements in OSS projects involve more security related commits than non-security requirements. Unlike previous external resources, CVE identifiers (CVE IDs) which are assigned by CVE Numbering Authorities (CNAs) and used to discuss and share information about a unique software vulnerability, is a strong indicator of security requirements. That means if a requirement contains a CVE ID, it is a security-related requirement. Therefore, we provide the following hypothesis for CVE IDs:

*H Security Commits: Requirements in OSS projects involve any CVE IDs shall be considered as security requirements.*

In summary, we manually analyze three open source projects. Based on our observation, we choose 6 meta-data heuristics which have potential to distinguish security and non-security requirements. We represent the rationales why we choose those meta-data heuristics/metrics of the machine learning algorithm as hypotheses. In the next section, we will introduce machine learning methods that we use to build security requirements identifying models.

### **3.2. Linear Classifier Based Approach:**

In this section, we describe the general architecture of our linear classifier-based approach. Our approach contains five parts: Information Preprocessing Component (IPC), NFR Classifier (NFR-C), Regression Models (RMs), CVE ID Detector (CID), and Linear Classifier (LC). Figure 3.3 presents the framework of our linear classifier-based approach.

Input requirements are processed by IPC to extract information of each metric described in Section 3.1.2. Then we feed this information to four regression models



(Comment Complexity Regression Model (CRM), Stakeholder Complexity Regression Model (SRM), Security URLs Regression Model (URM), and Security Commits Regression Model (CiRM)), NFR-C, and CID.

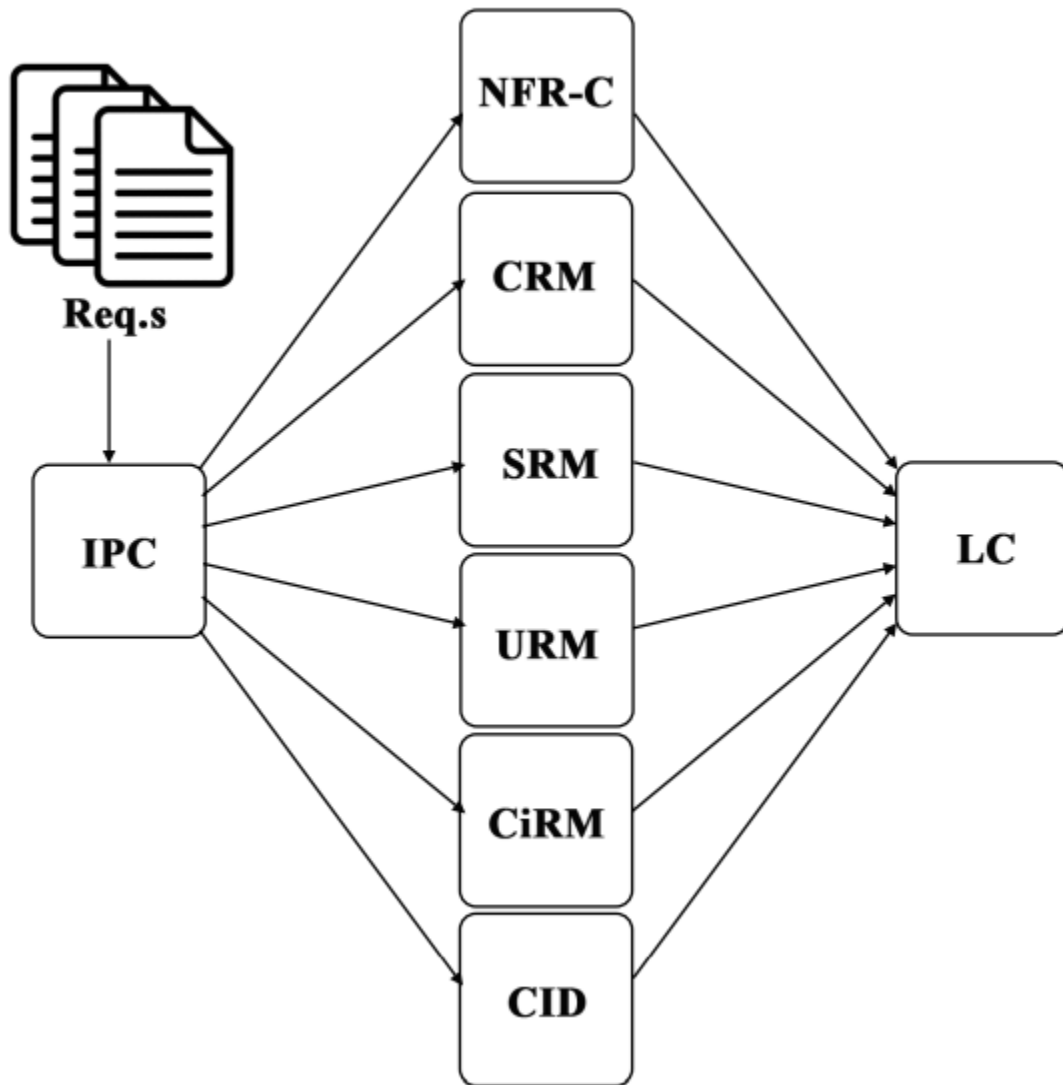


Figure 3.3: Linear Classifier Based Approach Architecture: NFR-C stands for NFR Classifier

For every requirement, each regression model will generate a weight between 0 to 1, which indicates the likelihood of this requirement being a security requirement. At the

same time, CID will detect whether this requirement contains a CVE ID. If there is a CVE ID in this requirement, the weight of it is 1; otherwise, the weight of it is 0. For NFR-C, it directly reuses the approach from [7]. Finally, a linear classifier is used to generate a linear discriminant function which is an optimal way to aggregate weights from NFR-C, CID, and all RM's. The result of the LC then is used to classify whether this requirement is a security or non-security requirement. We briefly describe the regression model and the linear classifier in the following subsections.

### *3.2.1 Logistic Regression Model*

Many classification techniques, such as J48 decision tree, Random Forest, Naive Bayes, Bayes Network, and Logistic Regression, have been widely applied to build security identification models [17, 26]. Lessmann et al. [26] investigated 17 classification techniques and reported that there is no significant difference in performance between those techniques. Same as [17], we used binary logistic regressions to identify security requirements.

Binary logistic regression computes the probability of occurrence of an outcome event from given independent variables by mapping the linear combination of independent variables to the probability of outcome using the log of odds ratio (logit). An open-source cluster-computing framework Apache Spark was used to build our regression models.

### *3.2.2. Linear Classifier*

A linear classifier is a statistical classification technique. The goal of it is to use an object's feature values to identify which class it belongs to. It achieves this goal by making

the classification decision based on the value of the linear combination of feature values [27].

The discriminant function of linear classification is shown in Equation (1):

$$f(\vec{\omega} \cdot \vec{x}) = \sum(\omega_i x_i) \quad (1)$$

where  $\vec{x}$  is a real vector of feature value, and  $\vec{\omega}$  is a real vector of weights for each feature-values. The weights vector  $\vec{\omega}$  is learned from a set of labeled training 320 samples by applying linear discriminant analysis (LDA).

Linear classifier is widely used in software engineering field, such as NFR identification [11] and CVS issue status prediction [28]. The major advantages of the linear classifier are its simplicity and time efficiency. Therefore, we believe linear classifier could be a good candidate for our approach. To our best knowledge, this research is the first one which applies linear classifier to solve the security requirements identification problem. Therefore, the following two fundamental research questions shall be answered: First, the function  $f(\vec{\omega} \cdot \vec{x})$  maps all values above a certain threshold to one class and all other values to the second class.

Therefore, the first research question is:

- RQ<sub>1</sub>: What is the optimal threshold for linear classifier when identifying security requirements? Second, in theory, the linear classifier can benefit from the optimal combination of different features, thus, it can achieve higher performance. However, to what extent the linear classifier can improve the performance of security requirements identification is not clear to us.

Therefore, our second research question is:

- RQ<sub>2</sub>: Can the linear combination of feature values achieve better performance than NFR-C and individual RMs?

### 3.3. Measurement and Results Analysis

#### 3.3.1. Measurement:

For evaluating binary identification models, we used recall, precision, and F-measure. Recall is defined as the ratio of correctly identified security requirements to actual security requirements:

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

where TP refers to true positives and FN refers to false negatives. Precision is defined as the ratio of correctly identified security requirements to all detected security requirements:

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

where FP refers to false positives. F-measure is a harmonic mean of recall and precision.

$$F_{\beta} = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \quad (4)$$

Failing to identify and realize security requirements will lead to a number of serious problems in software systems. Thus, we are more concerned about recall rather than precision. F<sub>2</sub> is a widely used measurement which fits our purpose.

### 3.3.2. Results Analysis:

To answer research questions, we applied 10-fold strategy to test our approach. That is, for each project, we randomly divided all the requirements into 10 subsets  $S = \{s_1, s_2, \dots, s_{10}\}$ . We used labeled requirements in  $S - \{s_i\}$  as input to train the classifier. Then we used this trained classifier to identify security requirements in  $s_i$ . All the results reported in this section are the average values of 10 runs of testing. Notice that we did not balance the training and testing sets (i.e., the numbers of security and non-security requirements may not equal to each other). Actually, from Table 3.1, we know that security and non-security requirements are not balanced in the real world. A small set of security requirements can completely describe stakeholders' security needs.

The output of linear classifier is a ranked list of the weighted requirements. The weight indicates the possibilities that a requirement is a security requirement. If the weight of a requirement is higher than certain threshold, linear classifier will mark it as a security requirement. In order to answer RQ<sub>1</sub>, we tested the performance of our approach via increasing threshold  $\theta$  by 0.1 from 0 to 1. The performance of our approach under different thresholds is shown in Figure 3.4.

The recall curve (Figure 3.4 (a)) shows that our approach can achieve 100% recall while the threshold is less than 0.3. While the threshold is greater than or equal to 0.5, the recall values of three projects decrease as the threshold increases. The significant decrease occurs while threshold increases from 0.9 to 1.0. This phenomenon indicates that many false positives occur at the top of the ranked list. We conjecture that there are two possible reasons. First, it may be due to that some non-security requirements have similar characters

with security requirements, we need other strong security indicators to distinguish them from security requirements. Second, it could be also caused by incompleteness of answer sets. We need to improve the quality of the answer sets for the future research.

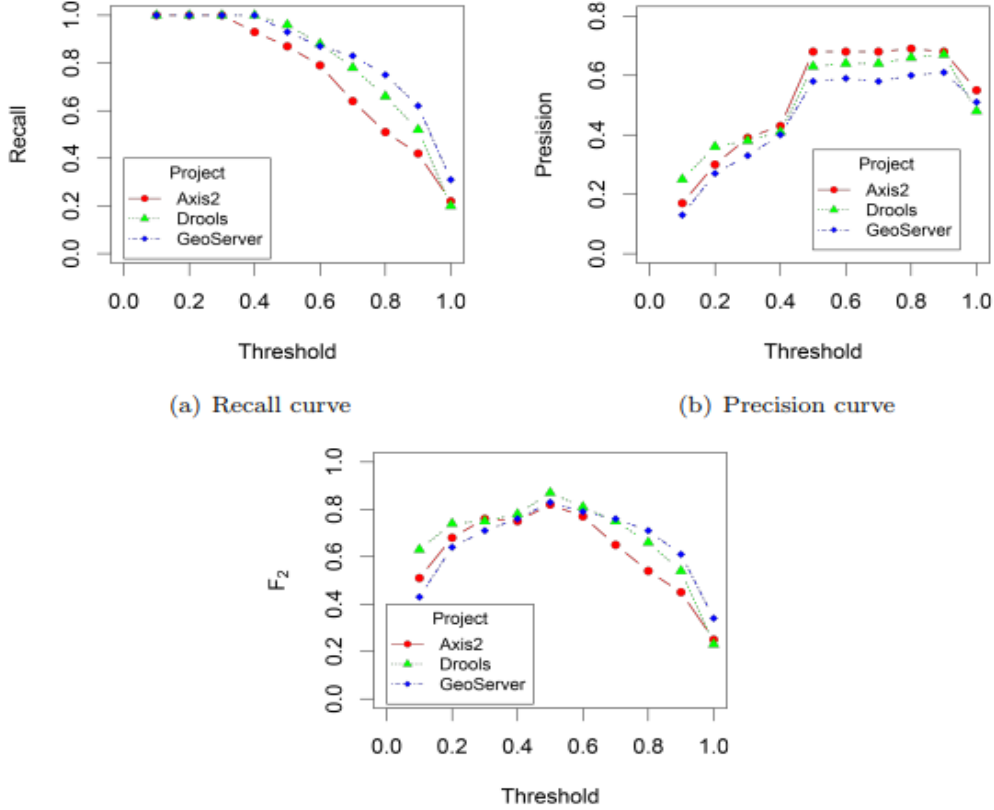


Figure 3.4. Linear classifier-based security requirement identification approach performance under different threshold.

For the precision curve (Figure 3.4 (b)), the precision values of three projects are asymptotically stable at an equilibrium point (i.e., threshold equals to 0.5). Additionally, in the F2 curve (Figure 3.4(c)), our approach achieved highest F2 values while the threshold equals to 0.5 in three projects. Based on these observations, we believe that 0.5 is an optimal threshold that can be used in our approach to classify security and non-security requirements. To answer RQ2, we compare the performance of linear classifier

with NFR classifier and four regression models. Like [17], in four regression models, a requirement is classified as a security requirement when the outcome probability is greater than 0.5. We do not include the performance of the CID. The main reason is, even it can achieve 100% precision, its recall value is very low, around (3% to 10%). The main reason could be that a few of security requirements are related to CVE IDs.

Table 3.4: Comparison between NFR-C, RMs, and LC: R refers to recall, P refers to precision, bold values are highest values in their columns

Approach	Axis2			Drools			GeoServer		
	<i>R</i>	<i>P</i>	<i>F<sub>2</sub></i>	<i>R</i>	<i>P</i>	<i>F<sub>2</sub></i>	<i>R</i>	<i>P</i>	<i>F<sub>2</sub></i>
NFR-C	0.59	0.28	0.48	0.56	0.32	0.35	0.57	0.19	0.40
CRM	0.74	0.19	0.48	0.78	0.23	0.53	0.68	0.14	0.39
SRM	0.66	0.19	0.46	0.75	0.26	0.54	0.49	0.11	0.27
URM	0.53	0.32	0.47	0.63	0.39	0.56	0.51	0.21	0.40
CiRM	0.47	0.67	0.50	0.52	0.61	0.54	0.44	0.53	0.46
LC	<b>0.87</b>	<b>0.68</b>	<b>0.82</b>	<b>0.96</b>	<b>0.63</b>	<b>0.87</b>	<b>0.93</b>	<b>0.58</b>	<b>0.83</b>

Table 3.4 shows the comparison results. Among five individual heuristics models, NFR-C, CRM, SRM achieved relatively higher recall but relatively lower precision. For NFR-C, the main reason could be that, in OSS projects, stakeholders are more likely to describe the requirements' function. Therefore, very little security indicator terms are used in their descriptions. The lower precision of CRM and SRM may indicate that the complexity is not only related to the security but also other non-functional requirements, such as performance. On the contrary, URM and CiRM achieved relatively higher precision. This again demonstrated that URLs and commits are strong indicators for security requirements. However, the lower recall values also indicate that not all security requirements contain these two features. Therefore, other complementary features are

needed to improve the performance of these two features. The results also show that, linear combination of all features (LC) achieved the best performance in all three projects. It is noteworthy that LC does not only improve the recall, but also improves the precision. This again demonstrated that LC is an optimal way to combine all features.

### **3.3.3. Discussion and Improvements**

We summarize the contributions of our study as below:

- Linear classifier-based approach:

We apply the linear classifier to identify security requirements. The results show that combination of six features can achieve average 92.31% recall and 62.94% precision on three OSS projects. One advantage of this approach comes from the simplicity of linear classifier. Compared to other machine learning methods, such as Naive Bayesian [29] and artificial neural network (ANN) [30], training and testing linear classifier are cost-efficient. Another advantage is that our approach is automated. We'd like to point out that we automated extracted features used in our approach by using JIRA REST APIs<sup>9</sup> and GitHub REST APIs<sup>10</sup>, except for URLs. It greatly reduces the proportion of the manual work, therefore increasing our approach's efficiency. For security URLs, automated topic analysis methods like Latent Dirichlet Allocation (LDA) can be used to identify them in the future.

- Threshold testing:

We applied our approach to three datasets. Through systematic analysis, we suggested that 0.5 is an optimal threshold for our linear classifier-based security requirements identification approach. Our experiments are then based on our suggested



threshold. The results reported in Table 3.4 show that our approach achieved better recall and precision (i.e., average 92.31% recall and 62.94% precision), compared to the state-of-the-art approaches (i.e., 82.8% recall and 7.1% precision in [7], or 70.48% recall and precision level is 67.35% in [13]). Again, Kurtović and Maalej [11] can achieve the highest precision (i.e., 90.0%) at similar recall level (i.e., 90.1%), however, their approach is not suitable for OSS 435 projects.

However, there are several main limitations in our study:

First, one major limitation comes from answer sets generation method. In order to increase the efficiency, we applied keywords searching to retrieve candidate requirements first. Two experts then only analyzed candidates to find security requirements. We acknowledge that pre-retrieval will cause missing security requirements, thus leading to bias in method evaluation. In addition, two experts may make incorrect decisions while judging security requirements candidates and also lead to missing security requirements. However, the Cohen's Kappa analysis result increases our confidence that two experts only put security requirements they feel confident into the answer sets. We argue that only using correct security requirements to train the models may help to increase the quality of our models and therefore increasing the automated security requirements detection. Second, like we discussed in Section 3.3.2, meta-data heuristics used to measure requirements complexity may also relate to other non-functional requirements. In the future work, we need to find other complexity metrics with the hope that they can improve the precision without decreasing the recall. In fact, recent research done by Dekhtyar and Fong [30] uses requirements' semantic feature (i.e., word-embedding) to build security identifying

models. Since our models use meta-data features of security requirements, we consider the effects of more semantic metrics in future work. Finally, we only test the linear combination of features by using linear classifier. Other linear combination methods like Naive Bayesian [29] can be tested in the future. In addition, non-linear classifier like kNN also can be used to build models in order to further improve models' accuracy. Dekhtyar and Fong [30] successfully applied ANN to identify security requirements. However, features used in our approach are not included in Dekhtyar and Fong's ANN model [30]. Training ANN model with our features could be a promising way to increase model's accuracy.

In this chapter we have considered the effects of meta-data heuristics in detail. In the next chapter, we consider the possibilities of more semantically enriched heuristics that can improve our ability to drill down all security requirements in a project.

## CHAPTER 4: INTRODUCTION TO SEMANTIC HEURISTICS

In the previous chapter, we have seen how security requirements are hidden implicitly and how the NFR classifier proposed by Cleland-Huang et al. [7] and meta-data heuristics like comments complexity, stake holder's complexity and external resources can be used to uncover such security implications. In this chapter, we focus on identifying the implications in the security requirements in a deeper level. This chapter deals with the possibility of engineering the classified security requirements to make them more operable and to check the effect of interdependencies.

The previous chapter clearly explains that the Machine Learning approach does not check the inter-relation between different requirements. To check for inter-relations and to study the effects of environment on the requirements, we scale down from Open Source Projects to Web applications. From our present results, we can clearly classify whether a requirement is security related or not. But, the result is just based on that particular requirements description, comment complexity, stakeholder's complexity, and external resources involved with that particular requirement. At the end of the previous chapter, we have discussed about how we consider the semantic aspects of a system in addition to the meta-data features that have been considered already. Another important aspect that could affect the security of the requirement is the system in which they are being implemented, i.e., The security aspect in any system is not limited to the requirement but has to be extended to the environment in which these requirements would be implemented and the semantics of the requirement. It is always necessary to keep the integrity, confidentiality and availability of the environment intact.

## 4.1. Security Related Semantic Aspects

In this chapter, we explain the different semantic aspects of Open Source Resources such as front-end libraries and projects. Security is often embedded within the requirements in more than one indirect way which are harder to detect than simply looking for the security indicators or using meta-data heuristics which may differ from project to project. This can be explained using different samples as mentioned below.

### 4.1.1. Conditional Trigger

Conditional Trigger is explained as a situation where the fulfilment of a particular condition may lead to a security implication. An example for conditional dependency can be seen in the following requirement from the AXIS-2 projects issue tracking system.



#### Details

Type:	<span style="color: green;">+</span> New Feature	Status:	<span style="background-color: yellow;">IN PROGRESS</span>
Priority:	<span style="color: red;">^</span> Major	Resolution:	Unresolved
Affects Version/s:	1.7.0	Fix Version/s:	None
Component/s:	transports		
Labels:	None		

#### Description

User agents commonly apply same-origin restrictions to network requests. These restrictions prevent a client-side Web application running from one origin from obtaining data retrieved from another origin, and also limit unsafe HTTP requests that can be automatically launched toward destinations that differ from the running application's origin.

To overcome this issue ,we need to implement CORS Spec.

[1] <http://www.w3.org/TR/cors/>

Figure 4.1. Illustration for Conditional Dependency.

The illustration in Figure 4.1 depicts that unless the CORS support is enabled the client-side web application running from one origin cannot access the data retrieved from another origin i.e., CORS not only gives permission to the web application running on one origin to access certain resources from the server running on another origin, but also does not limit unsafe HTTP. Since this process involves blocking and sharing of data between two different origins and not limiting the unsafe HTTP requests, it can be considered as a security requirement. The security implication here depends on whether same-origin restrictions are applied to network requests or not.

Conditional triggers are often seen to occur in web applications. This can be seen in one way as a permission methodology as their occurrence spans monitoring origins in the network to monitoring the format of the data being sent.

#### **4.1.2. Requirements Centric Implications**

These security requirements are purely and solely based on the particular requirement at hand. These can be classified and categorized as per the requirements mentioned in the previous chapter. These requirements can be classified using Cleland-Huang et al. proposed approach of using specific keywords or the meta-data heuristics like complexity (complexity of comments, the stakeholders involved and external resources) proposed in the previous chapter to a certain extent only as they do not capture the underlying meaning of the sentences described in the requirement.

In reality, we often notice that security may not only depend upon a single keyword. There can be a combination of terms which might affect the entire meaning of the requirement. This is one of the drawbacks of our heuristics considered in the previous

chapter, as we do not handle the meaning of an entire sentence. An example of such situation can be seen in Figure 4.2. From the figure, it can be seen that one of comments clearly specifies that “Anyone who has access to the server can see the source code”, on using the security indicator keys, this may not look like a security vulnerability. But, on analyzing the meaning, it is pretty clear that any unwanted access to the servers may lead to source code leakage. This can turn out to be a huge security threat if the source code or access to server gets in the hands of attackers.

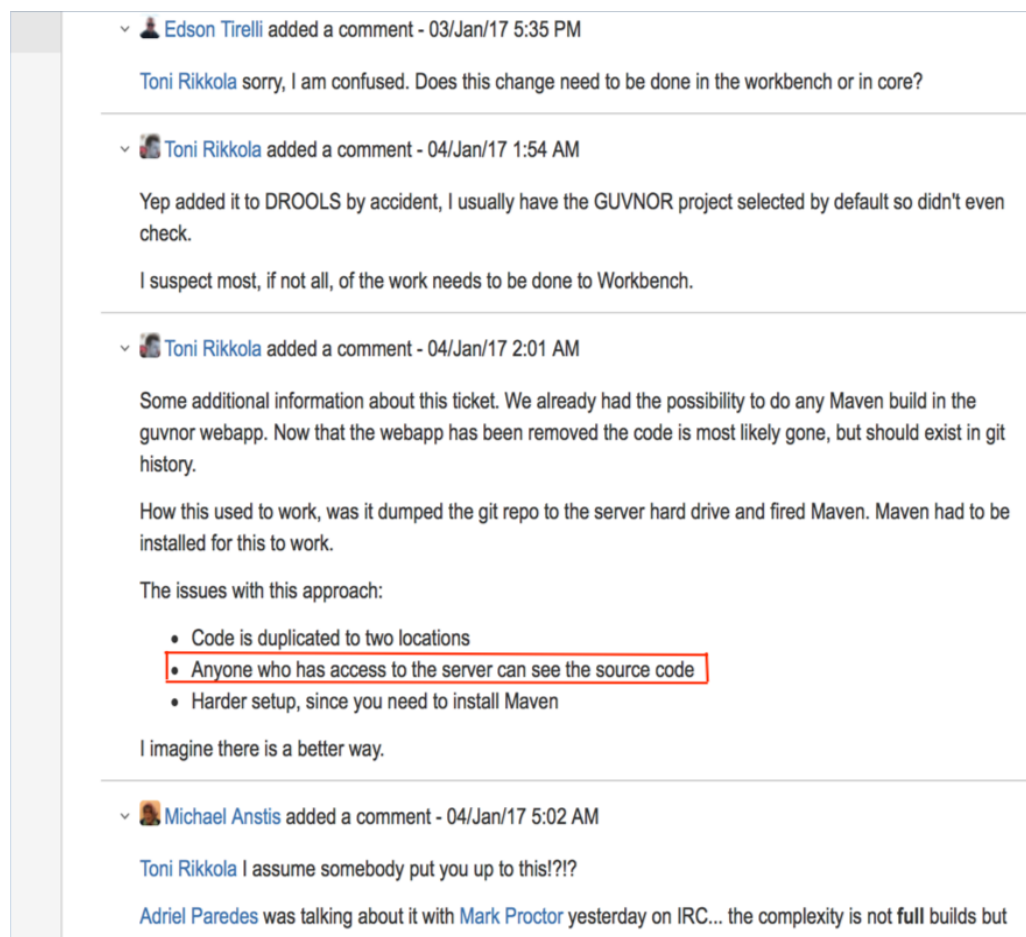


Figure 4.2: Illustration of Requirements Centric Implications

Our classification in the previous chapter fails to acknowledge such situations. In [41], Niu et al. proposed a method to check the lexical affinity between words in a sentence.

Techniques like this can capture the essence of the sentences by finding the combinations that may have the most value in terms of security based on the lexical affinity present in the sentence. A combination of words that are most often used together to cause a security implication could be considered in addition to our security indicators in the previous chapter.

#### 4.1.3. Depreciated Dependency

The depreciated dependency is a behavior among the requirements that can be described as the scenario where the deletion of a requirement or the related resources causes a security implication.

It is observed that in many other cases, dependency of web applications on any third-party resources leads to security implications. This can be explained using the following illustration in Figure 4.3.

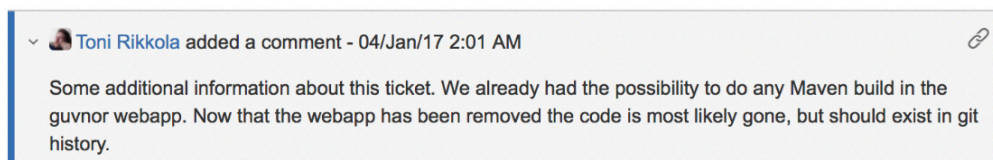


Figure 4.3. Illustration for Depreciated Dependency

As we can see in Figure 4.3, as a part of Drools project. The issue deals with using maven for compiling the workbench projects. In this issue, after a good amount of discussion it was seen that although the web application being dealt with was deleted, the code remained in the git history. This marks as another example of depreciated dependency. Here, sharing the code with third party resources has triggered the security

implication, only when the web application was deleted but not the resources shared with the third party.

This scenario is not considered in the external resources topic of our previous chapter, so we could consider an additional heuristic where the third-party repositories mentioned to be linked with the web applications are scanned for any traces even after deletion of the application or resources.

#### **4.1.4. Derived Implications**

In the context with web applications, 80% of the code for web applications is taken from libraries and frameworks, and about one fourth of library downloads have known vulnerabilities [38]. Until now, we have discussed about how a requirement might have security implications. Now, we move on more deeply to see how the environment in which the requirements work affects the security aspects. Though difficult, we can impose control on the security practices and aspects of the implementation to some extent. But, with the amount of open source libraries that are available and the advantages in terms of time in reusing those libraries, every organization, big and small, every developer group, amateur to advanced are using the available libraries. It is clear that open source libraries have become an integral part of many projects being built along with application software's tools and documentation.

As mentioned earlier, the concept that most of the web applications run on the fact that the code uses already existing frameworks and libraries is very common. As a matter of fact, it is considered as a good practice to have code which can be refactored and reused to reduce the effort and to maintain common methodology among working organizations.



Though this process is very helpful in saving time and effort, many times the libraries and frameworks being used may be affected with security vulnerabilities. Using these affected resources in our projects may lead to making our projects prone to these security vulnerabilities. Such security vulnerabilities derived from other libraries or frameworks into the requirements can be called Derived Security Implications.

In this chapter, we try to showcase not only how open source resources/libraries affect the web application leading to an issue of open source security but also how the derived implications can be used with our classifier results to check if we can correct some of the wrongly predicted labels.

To understand the Derived Security Implications in detail, we can consider different dependencies like package dependencies, version dependencies etc. that are supported in web applications or open source projects in general. However, in the following work we focus on the working of packages and its dependencies as an open source resource.

## **4.2. Working of Packages**

As discussed above, the derived implications mainly deal with the hierarchy of both libraries and packages. In the context of web development, libraries and packages are treated very close to each other. The simplest way to describe this is, libraries relates to a block of code that can be reused at different places of different projects, whereas the working of package enables us to distribute a set of libraries that it contains. Therefore, it is safe to claim that a package contains a set of libraries which can be used upon downloading the package.

In the open source world, the packages available are rigorously updated and new packages are constantly added. Although improvisations have the advantages of the latest releases in terms of vulnerabilities, bug fixes and addition of functionalities, they also raise issues of compatibility with its dependent packages, implementation and technical lags.

Packages include capabilities which are a description of the functionality that they provide. There can be packages which have the same functionality that they provide. However, when a particular project tries to download more than one package with exact capabilities, it results in conflict.

#### **4.2.1. Package Dependency**

Package Dependency is an important mechanism followed in the working of packages. This is one of the most used software reuse methodologies that is followed by all the developers in not only web application development but also in every other automated research.

Package Dependency is mainly having one package which depends on the other packages. In a broad scope, package dependencies may look complex because of the connectivity and network that exists. But, in reality package dependencies most often are advantageous not only in saving the disk space but also in modularizing code. The modularized code is generally not only organized but also handy for developers when a new version of package has to be released. Modularized code is aimed to work with only a particular section of code without changing the functionality of others.

As specified in the previously, in ideal cases any change in one package does not interfere with the functionality of any other dependent package. But, in reality handling

package dependencies is a little trickier than that. Often, changes in one package cause incompatibility with its corresponding dependent packages. Also, it is important to remember that updated versions of one package do not update its dependency packages totally. We will discuss more about versioning and how package managers handle them in the later part of this Chapter.

In spite of the numerous advantages of package dependencies, these dependencies often experience security vulnerabilities. The security vulnerabilities in such package dependencies have a high impact because of the linkage between different packages. The vulnerabilities detected are mostly medium and high according to the findings by Decan et al. in their attempt to find the number of packages affected by vulnerabilities [40].

Now let us consider in depth the effects of package dependency. Consider the report by Snyk, a company analyzing software vulnerabilities in Node.js and Ruby packages. The report outlines the current state of open source security [39], [40]. Based on the analysis of 4,30,000 web applications, a final report leads us to results that about 3,31,100 web applications run on front-end libraries contain security vulnerabilities. This accounts for 77% of the total web applications that were in study. There have been a lot of discussions about the vulnerabilities caused by such open source libraries and many open source communities are working towards minimizing these vulnerabilities.

Analyzing such security vulnerabilities in open source libraries [40] not only tries to answer research questions about the number of packages that have security vulnerabilities, the amount of time that they stay active, and the discovery and resolution of such vulnerabilities, but also about the effects in dependent packages.

It is important to keep in mind in real-time applications that application not only extend external open source resources/libraries but also extend other packages involved in the same web application or external applications. Any vulnerability in one package would also be injected into the application or classes extending the affected package. This triggers a point of considering the effect of dependent packages for identifying security requirement.

To explain the concept of dependency more clearly, we can consider the npm (node package manager) package distribution. The existence of large number of packages definitely requires a package manager. npm is one such package manager which is used in many JavaScript and its framework related projects as it contains many libraries which can simplify the implementation of front-end functionalities in projects just like RubyGems for Ruby and CRAN for R. npm is one the largest package distributions consisting of about 6,41,128 packages. From [40], [42], [43], we also know that npm package distribution has higher dependency distribution of JavaScript packages than other packages. So, npm is a good package manager to consider for the study.

npm is a registry that is used widely because of the wide variety of packages that it handles. The description of the diversity of npm can be seen in Table 4.1. The table clearly explains how npm registry allows developer to update, delete, or add packages. The ratio of updates when compared to adding and deleting packages is high. Based on the type of updates that are made, the versioning pattern is set. The basic versioning pattern is major.minor.patch. This versioning is maintained according to the releases, i.e., the major section for the versioning is incremented when the releases are backwards incompatible,

while minor releases are incremented when they can be reversed backwards, patch attribute is incremented for bug fixes. Therefore, it can be inferred that npm provides clear versioning techniques.

npm is advantageous when compared with other package managers in terms of versioning. Many package managers expect the version of a package to be canonical. This requirement can be irritating to the developers when they know that the entire progress has been stopped because of a versioning problem and they would have to correct all the dependencies to continue the development and execution process. npm rescues the developers from such versioning issues as it allows a project to have more than one package version. This is not a problem in npm because it follows a system of every package having its own dependencies. This does not force all the package to have the same versions as separate tree structures of every installed package supports multiple versioning.

npm also provides clear categorization of packages. There are four types of categorizations provided. The first category included in npm is Popularity. These packages are generally the most downloaded packages. Based on the number of downloads it would not be wrong to tell that these packages are considered very helpful in development in terms of the functionalities that they possess. The next category is Quality. This category contains packages that are well documented with all the files necessary. These packages contain up-to-date dependencies specifications and readme files related to them. The third category is Maintenance. These packages have well versed developers working on them continuously to protect them against vulnerabilities and to improve the functionalities

provided by these packages. The final category is the Optimal. This category is a combination of all the previous categories.

Table 4.1: Overview of npm package manager.

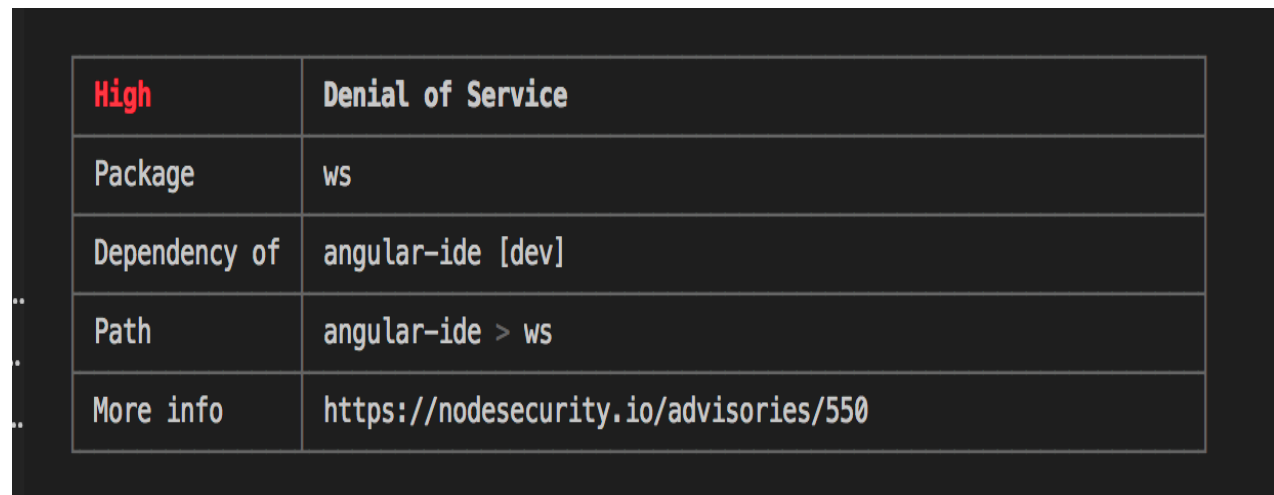
<b>npm package manager</b>	Number of packages involved	Approximately 6,00,000	
	Activities involved	Adding, Removing and Updating	
	Versioning	Yes	Major
			Minor
			Patch
	Categorization	Yes	Optimal
			Popularity
			Quality
			Maintenance
	Vulnerabilities	269 reports <sup>3</sup>	
	Packages depending on vulnerable packages	1,33,602	

npm has numerous advantages, but being a resource which contains packages accessible to everyone, it is not shielded from security vulnerabilities. From Table 4.1 we

---

<sup>3</sup> From the description of npm dataset in[40]

can see how more than 1,00,000 packages are indirectly or directly depending on vulnerable security packages. Therefore, the npm packages contain security vulnerabilities. Due to package dependencies these vulnerabilities can affect the dependent packages as well. In order to detect and know more about vulnerabilities in the packages, npm provides a method of using npm audit command in the project in which npm packages are loaded. This npm audit command generates a detailed report about the vulnerable packages and the degree of vulnerability as shown in Figure 4.4 (a) and Figure 4.4 (b),



The image shows a terminal window with a dark background. It displays an npm audit report for a 'High' severity vulnerability. The report is structured as a table with two columns. The first column contains labels: 'High', 'Package', 'Dependency of', 'Path', and 'More info'. The second column contains the corresponding details: 'Denial of Service', 'ws', 'angular-ide [dev]', 'angular-ide > ws', and 'https://nodesecurity.io/advisories/550'. To the left of the table, there are three lines of dots: '..', '..', and '..'.

<b>High</b>	<b>Denial of Service</b>
Package	ws
Dependency of	angular-ide [dev]
Path	angular-ide > ws
More info	<a href="https://nodesecurity.io/advisories/550">https://nodesecurity.io/advisories/550</a>

Figure 4.4 (a). npm audit report containing High severity security vulnerabilities

The path information in Figure 4.4 (a) and Figure 4.4 (b) shows the path where the vulnerability exists and the dependent packages. The path description contains the package name followed by the dependent packages followed by the security vulnerability.

<b>Moderate</b>	<b>Prototype pollution</b>
Package	hoek
Dependency of	@angular-devkit/build-angular [dev]
Path	@angular-devkit/build-angular > node-sass > node-gyp > request > hawk > sntp > hoek
More info	<a href="https://nodesecurity.io/advisories/566">https://nodesecurity.io/advisories/566</a>

Figure 4.4(b). npm audit report containing Moderate severity security vulnerabilities

The npm report shown in Figure 4.4 (a) and Figure 4.4 (b) clearly shows the degree of severity, the type of severity, the package containing the vulnerability, and the dependency package. The report also generates information about the packages that need serious fixes in order to avoid severe threat to the projects. We can also notice the most useful path information in the report which leads to the affected package.

#### 4.2.1.1. Package Dependency Graphs

A useful demonstration of npm package dependency is shown through the graphs created using npm package manager using the tool in [50]. In order to create these graphs npm documentation is used. The official npm documentation provides details like the version of the package, the number dependencies it contains, the number of dependents and a list of all dependents and dependencies.

Using the information provided in the npm documentation, the meta-data is resolved and used to build a graphical model. To analyze Figure 4.4 (b) better, let us see how the graphical representation of the npm dependencies are formed.



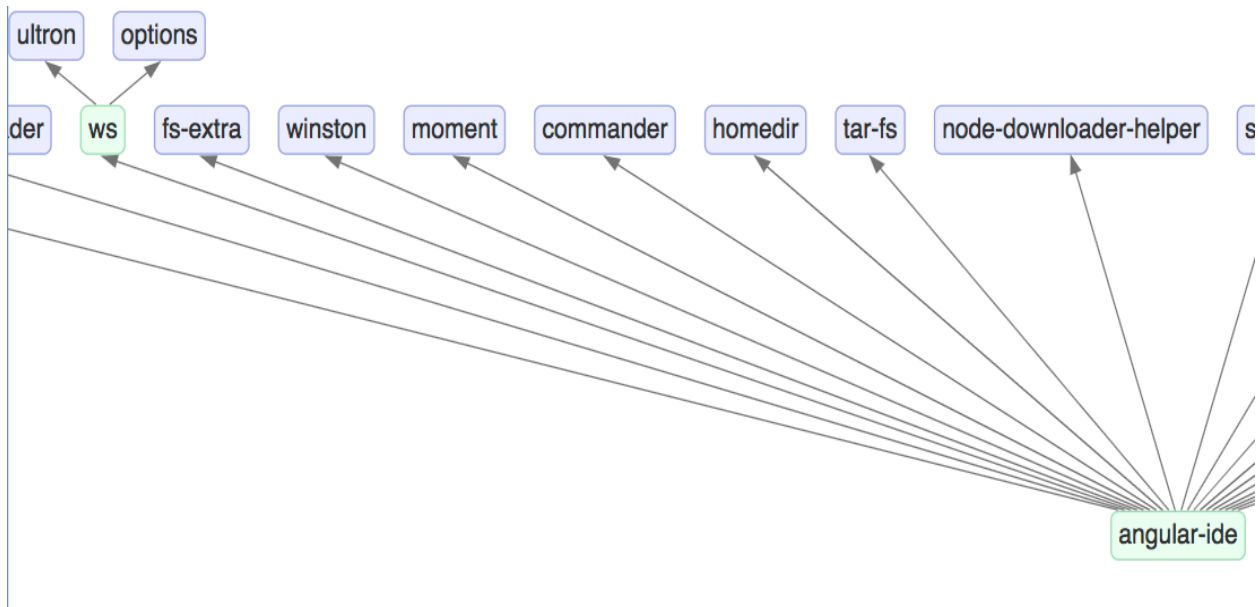


Figure 4.5 (a) Package Dependency graph while downloading ws

Figure 4.5 (a) shows how the package angular-ide has multiple dependencies. Therefore in order to download package ws the dependency packages need to be downloaded.

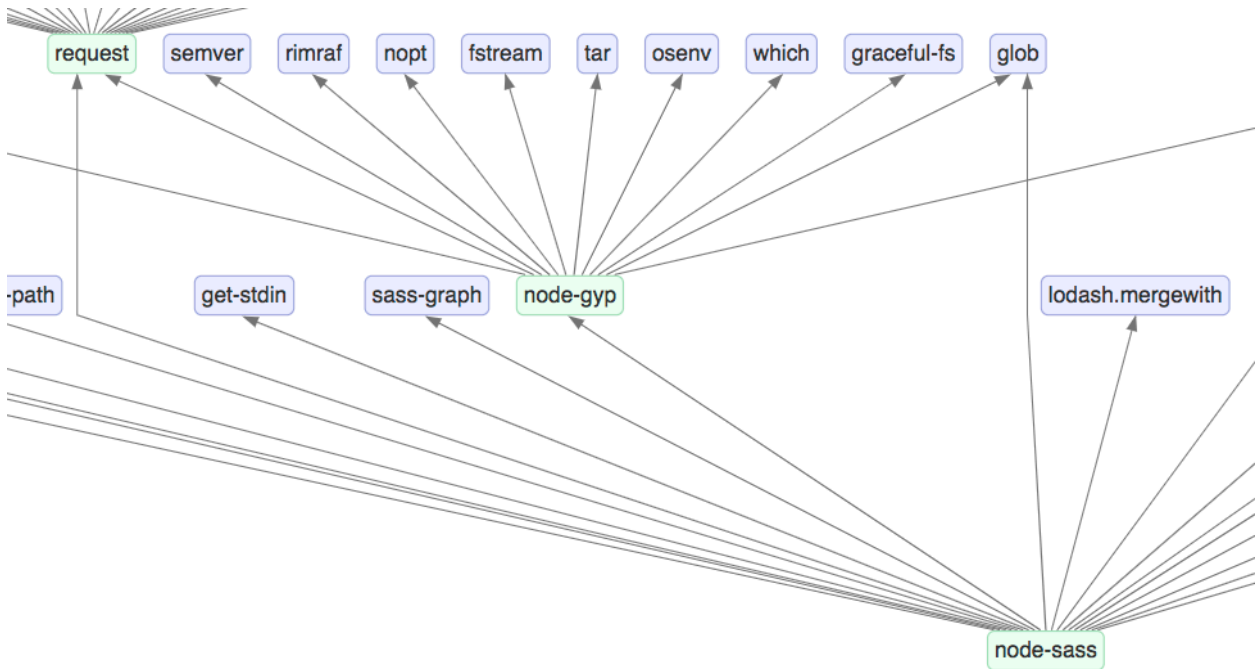


Figure 4.5 (b) Package Dependency graph while downloading hoek

The Figure 4.5 (b) shows dependency graph of different packages that are involved a section of the path involved in downloading hoek. The figure shows only a part of the actual dependency graph generated. Often, due to the high number of dependents and dependencies, the graph size is extremely large with a lot of interconnected nodes. In Figures 4.5 (a) and 4.5 (b), we only consider the dependencies of the vulnerable package that were detected in Figures 4.4 (a) and 4.4 (b) respectively.

Another useful npm command is the npm list. This will help in listing all the npm packages that are installed. It gives the name of the package, the version details, and the source tree of the npm package as shown below in Figure 4.6.

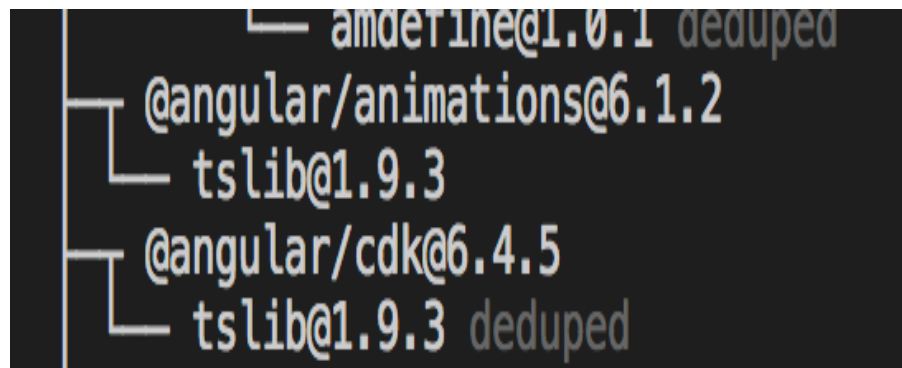


Figure 4.6 npm package list using npm ls/ npm list command

Another form of derived security implication can be attacks like SQL Injections on web applications. SQL Injections are often SQL statements that are given to web applications as input in order to start an attack. This is one of the popular hacking techniques. In web applications, we often notice validators on the inputs, validations on both front end and server side. These validators are used so that the web application can be protected from the malicious attacks by intruders. The intruders use information passed from input parameters in web forms, cookies to start attacks on the database servers

connected to web applications. Absence of validators in this case would lead to serious consequences like misuse of web application or leaking of confidential information. Such SQL Injections are treated as derived security implications as they are not originally associated with application until the point when an intruder injected them to the application.

In recent times, much research is being conducted to study the similarities in the patterns of the code that is associated with SQL Injections. Sources for these studies are mostly the HTTP methods, configurations, custom functions and HTTP wrapper methods as mentioned in [44]. According to [44], SQL Injections along with Cross Site Scripting, Memory Corruption, Cross-Site Request Forgery can be considered as one of the underlying details of CVE which has been discussed in the previous chapter. We discuss more about how a web application is affected by such techniques in the next chapter.

### **4.3. Potential Heuristics**

Although in this work, we mostly focus on the how package dependencies act as a semantic heuristic, we also propose a few extra heuristics apart from the ones considered in Chapter 3 and describe how they have a potential to affect the results produced by our linear classifier. The following additional heuristics are introduced,

- i. Considering the combination of words in addition to the security indicators terms. The combination of words is considered using the technique mentioned in [41]. As we know that the issues in the issue tracking systems are generally written in natural language, the associated comments are also in natural language. To get the combination of words which may have the

most impact, we use the technique used in [41] to extract Functional Requirements Profiles. This technique captures the lexical affinity between words of a sentence which can be advantageous in situations where we want to handle requirements-centric implications that are introduced in the earlier stages of this chapter.

On any mention of deletions or depreciations, the associated third-party links like Git are checked to make sure if the respective changes have been reflected there or have been ignored.

Traces for SQL Injections in the code associated, code snippets or any discussion about such intrusions in comments or issues mentioned are checked. All these comments and descriptions would be in natural language which is why we think that considering the lexical affinity would be more precise than just considering the security indicators.

- ii. Commands such as npm audit are used to check to see if a package is vulnerable. Any dependent packages in the same web application are flagged too. Therefore, considering the packages being downloaded in order to fulfil a requirement can help us to track the dependencies of the package and check if the package has any security vulnerabilities. The severity of the vulnerabilities in the package being downloaded and the path attribute from the npm audit reports can be used to consider the security implications imposed in terms of packages that are generally ignored by our meta-data heuristics.

We now have a clear understanding of how these different semantic heuristics can be used to create a potential improvement in the results produced by our linear classifier. However, in this work we mainly focus on the package dependencies and their effects on our classifier.

In the next chapter, we consider a case study which explains how applying package dependency semantic heuristic can improve our classifier and overcome the limitations faced by our classifier to detect security requirements under certain circumstances.

## **CHAPTER 5: CASE STUDY**

### **5.1. Introduction**

In the previous chapters we have not only seen how meta-data heuristics affect the classification of security requirements, but also picked out the semantically enriched heuristics which could affect the security requirements. Until now, we have only studied the type of heuristics that could be considered, but not the correlation between them. To have a deeper understanding of how semantically enriched heuristics affect the meta-data heuristics, in this chapter we consider a corporate web application as a case study.

#### **5.1.1. Background**

In the corporate world we notice that organizations generally tend to customize the applications that they use within the organization. While some organizations use the openly available applications like Skype or Hangouts, others like to have their own customization based on the type of usage and work. Similarly, we are going to discuss a communication-based project to test our heuristics. The web application we are considering for this case study is a mass emailing system which is focused to serve the internal communication within an organization.

The emailing system is built for a medicare oriented organization which wants to have a record of all the mass emails sent to employees as their email information contains information to be sent to a specific set of people working in the organization. Since the emails sent within this organization contain sensitive information and the organization wants to maintain a detailed database about the emails and its contents, the organization

has chosen to build their own application instead of using a software like listserv which is already available.

The organization contains of about 15,000 employees. However, in the pre-release versions only 500 employees have been selected access to the application. These can be seen as ‘early adopters’ of the intended software [48], [49]. The developer team consists of 3 developers. These statistics contribute to get an understanding of the number of stakeholders involved. The initial requirements that were mentioned for this project were shared with the developers and reviewed and revised several times. Any issues in the development or functionality are reported to the developers using emails, meetings, and the issue tracking systems comments section. These issues are resolved in a timely manner (i.e., mostly with 7-10 days) and new features are added by considering the inputs of the stakeholders and contemplating the pros and cons of the features. The technologies on which this project was built are Spring Boot for building the back end and Angular for building the front end. The development process involved using of BitBucket as the external resource to push and pull the commits of the code. A record of all the code commits is maintained in the issue tracking system used by the application as well.

In the context with the technologies, it would be helpful for us to know that both Spring Boot and Angular have different versions. Every version has new features and enhancements. However, their foundations remain open source. The underlying layer for Spring Boot is Java. On the other hand, Angular uses libraries from npm packages for its functionalities in projects. Therefore, we can assert that Java and npm are the foundation for this project, while Spring Boot and Angular are the building blocks for the email

application under study. The project is also a part of third-party resources like BitBucket and Git as a gateway for tracking the code changes during development. The platform used to build this project was Spring Tool Suite (STS) and Visual Studio Code.

### 5.1.2. Requirements

Before developing any project, it is very important to have a clear understanding of the requirements or the project itself in order to assess the costs, quality, and schedule[51]. Software requirements engineering is considered as a “soft” human-centered activity which explains the systems whose core is software [52]. Following the basic principles of software engineering, it is pretty clear that the clearer the requirements, the more advantages of reduced cost and completion time.

The functionality of the web application used in our current case study is that the application would be a secure application using user authentication before letting the user access the application. The application should give the user access to certain default templates. It should also let the user create new templates and save them for future use. The users should be able to modify the templates and send out emails by using the templates. The information related to any email sent or any new template added should be saved to the database.

A few requirements from the web application we are considering are shown below:

Table 5.1: Subset of requirements that were framed for our emailing system web application

Issue ID	Category	Title	Description
ESYS0015	Feature Request	Authentication support needed	Authentication has to be based on credentials given by company to the user. The user would have to enter the company email and password to login to the utility.



			This can be done using Spring Security Framework and LDAP Authentication where LDAP acts as a central repository for the user information and the web application can connect to it for authentication.
ESYS2124	Enhancement	Manage project using pom.xml	The application has to use Maven in order to build the spring boot application, for representing dependencies and for documentation
ESYS1341	Feature Request	Adding CORS support	User agents commonly work on same origin to network requests. These conditions make the client-side web application to run from one origin from obtaining data retrieved from another origin. So, we need to apply CORS support.
ESYS6132	New Feature	Encrypt sensitive information in the email to allow it to read by only authorized users.	The email utility can send kick-start URL's and passwords for new recruits in the company. This requires the right passwords to be visible to only the right users, for which the URL's and passwords can be encrypted.
ESYS7923	Enhancement	Allow users to save most commonly used recipients list file to the application.	Specify most used recipients section in mail options so that user can attach the recipients directly from application instead of their computer.
ESYS3211	Enhancement	Run security vulnerability check	The angular framework uses npm packages to build the front end. Before the packages are used, the npm packages can be audited using npm audit command to check for any security vulnerabilities in the package itself.
ESYS4542	New Feature	Introduce a preview option before sending out emails	The preview option has to show all the details including the emails from the parsed recipients file.
ESYS7372	New Feature	Show uploaded files to user on request and provide a delete option	To enhance ESYS7192, add a "show files" options which will let the user see all uploaded files that need to be attached in the

			email. The user should be provided with a delete option next to the file.
ESYS8781	New Feature	Create templates based on roles	The users should be able to create and save templates that they use most often. The templates created by one user should not be visible to any other user.
ESYS2787	Enhancement	Download form-data related packages to pass the form data to server.	To send the form-data onto the server, download form-data and its related packages.

The above table shows a sample subset of requirements defined by the organization to be implemented in the web application. There are a total of 138 requirements (feature requests, enhancements) in this project so far. In any case study, understanding the system is the key. Next, we look at the web application during the implementation phase for better understanding of our case study.

### 5.1.3. User Interfaces

In order to analyze any system, it is very important to know how it looks like and the software that is used to build the system. We now consider the front end of our web application and dig into the specifications that are involved in building it.

The web application that we are using in our case study most usually uses Angular CLI. The version of Angular CLI used is 6.1.3, with Node 9.3.0 on a Darwin x64 OS and Angular version 6.1.2. These versions of Angular are comparatively new, so they are expected to have dependencies with other versions. The web application involves installation of many other packages. A sample of the packages that are involved in the project and installed is shown in Figure 5.1. The versions of each of the packages involved are also mentioned in Figure 5.1.

Package	Version
@angular-devkit/architect	0.7.3
@angular-devkit/build-angular	0.6.8
@angular-devkit/build-optimizer	0.6.8
@angular-devkit/core	0.7.3
@angular-devkit/schematics	0.7.3
@angular/.DS_Store	2.7.2
@angular/cdk	6.4.5
@angular/cli	6.1.3
@angular/material	6.4.5
@ngtools/webpack	6.0.8
@schematics/angular	0.7.3
@schematics/update	0.7.3
rxjs	6.2.2
typescript	2.7.2
webpack	4.8.3

Figure 5.1. Sample packages installed for the web application.

The image shows a login page with a light gray background. At the top, a red error message reads "Your User ID and/or Password are invalid." Below this are two white input fields for "USER ID" and "PASSWORD". Underneath is a language selection dropdown menu labeled "SELECT A LANGUAGE" with "English" selected. A purple "SIGN IN" button is positioned below the dropdown. At the bottom, there is a checkbox labeled "enable accessibility mode".

Figure 5.2. Login page

Figure 5.2 shows how the web application is secured using an authentication system. The authentication for this particular project uses LDAP. Only the users that work within the organization and who have an id provided by the organization are allowed to login to the system.

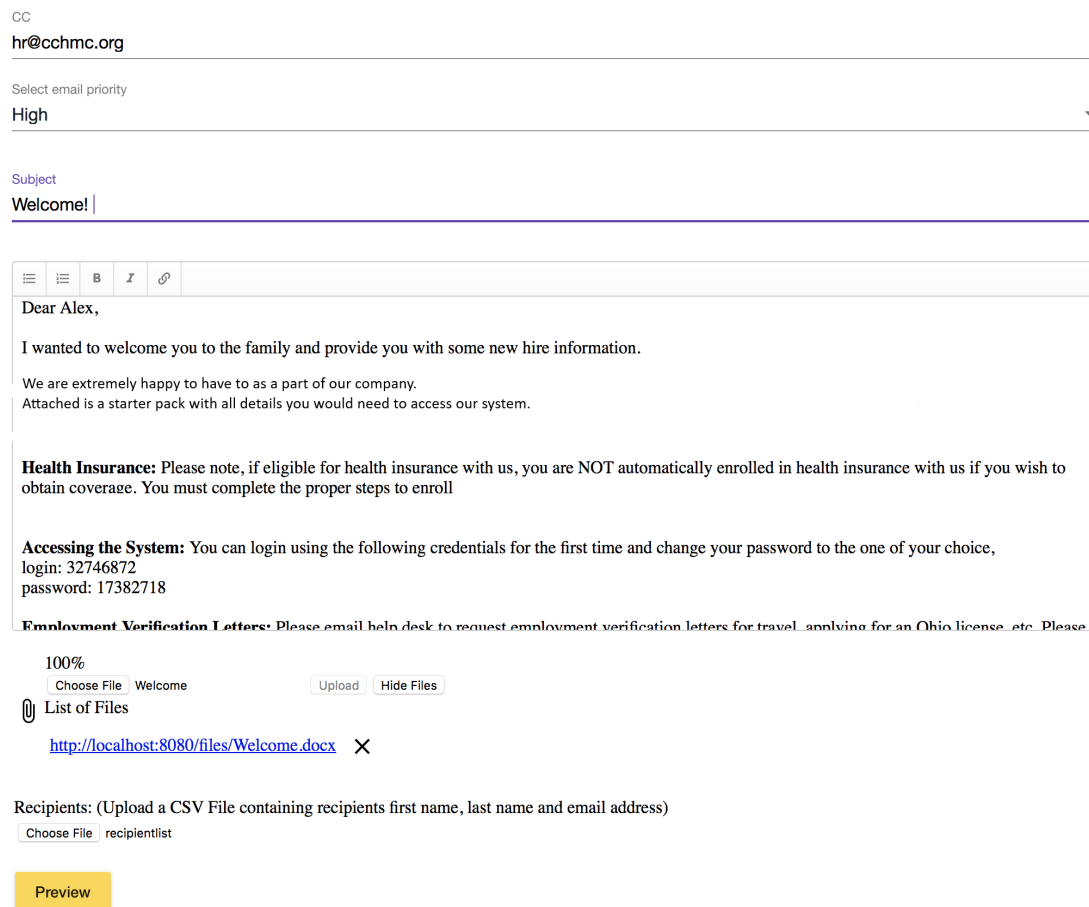


Figure 5.3 Interface of the web application

Figure 5.3 is an example of how a welcome template is being used to send initial set up details to new employees.

The above figure shows the preview of the email that is being sent along with the recipient emails from the parsed recipients file which is uploaded by the user. Now that we understand the basic look and feel of the application we are considering in our case study,

we shall move into analyzing this web applications requirements with the heuristics that we have proposed.

Another important front end library being used is Angular Material. Angular Material is a library that used in this project for beautification of angular components such as themes, buttons, and form fields. The version of Angular Material should be compatible with the version of Angular being used.

#### **5.1.4. Validators**

Any web application contains validations in order to avoid wrong or unwanted inputs. This helps in securing the application and maintaining consistency. In the current application, validators were placed in the information fields like bcc, cc, subject and body. Angular has a built-in ability to not allow inputs with special expressions in them. This is because SQL Injections mainly contain entering special expressions or queries in the input fields which mess up the backend when the input form is sent to the backend. Therefore, in the present scenario, validators are introduced in almost every web application for increased control and correctness in the application.

#### **5.2. Results and Analysis**

Using our case study, we evaluate the heuristics that we proposed in the earlier chapters and check if our heuristics complement or contradict each other. In order to do this, we consider the same sample of requirements that we have randomly chosen from a pool of requirements available for our case study as shown in Table 5.1.

In this chapter, we mainly try to address the following research questions:

*RQ<sub>1</sub>*: How do meta-data heuristics influence the classification of requirements in the web application under study?

*RQ<sub>2</sub>*: How do environment-oriented heuristics like package dependencies affect the security of requirements in the web application under study?

*RQ<sub>3</sub>*: How can we handle the false negatives obtained by our linear classifier?

Using the meta-data heuristics proposed in Chapter 3, we have already tested our heuristics for three open source projects: Drools, Axis 2 and GeoServer. While classifying the requirements in these three projects, we have already discussed and evaluated our classifier in Chapter 3. We now use the same linear classifier to classify the requirements of our case study.

On classification using the linear classifier built in Chapter 3 and using the requirements of our case study we obtained a recall of 0.82, precision of 0.67 and F-measure of 0.85. A deeper look of the classification results of our sample requirements is shown in Table 5.2.

The Table 5.2 describes the classification using our meta-data heuristics such as the comment complexity, the commits, and the number of stakeholders. The label 0 represents that the requirement is a non-security requirement, whereas the label 1 represents that the requirement is a security requirement. The other columns of the table show the manual analysis of the classification, i.e., the reason why the classifier may have marked a particular requirement with its respective label. It is to be noted that although we have done manual analysis of heuristics like stakeholders and comment complexity, we consider the

number of email conversations between the teams and the minutes of meeting for comment complexity in our current case study. The data gives an idea about how RQ<sub>1</sub> is handled.

Table 5.2: Classification and Analysis of requirements using meta-data heuristics

Requirement ID	Label	S. I	H <sub>cc</sub>	H <sub>sh</sub>	H <sub>URLs</sub>	H <sub>com</sub>
ESYS0015	1	✓	✓	✓	✓	✗
ESYS2124	0	✗	✗	✗	✗	✓
ESYS1341	0	✗	✓	✗	✓	✓
ESYS6132	1	✓	✓	✓	✓	✓
ESYS7923	0	✗	✓	✗	✗	✗
ESYS3211	1	✓	✓	✗	✓	✓
ESYS4542	0	✗	✗	✓	✗	✓
ESYS7372	0	✗	✗	✗	✓	✓
ESYS8781	0	✗	✗	✓	✗	✓
ESYS2787	0	✗	✗	✗	✗	✗

Let us consider one requirement from the Table 5.1, say ESYS2124. This requirement did not contain any security indicators defined by Cleland Huang *et al.* [7], there was not a lot of discussion on the requirements development, deployment or usage, the stakeholders involved are essentially only the developers, and there are no URLs included along with this requirement. However, the code commits include the pom.xml file that is dealt with in the requirement as the pom.xml file contains all the dependencies required for the function of the web application like the JPA repositories, the spring

security dependencies, the LDAP dependencies, etc. Therefore, any changes made to the pom.xml affects the functionality of the web application, which is why any changes in pom.xml code are committed to the external resources (BitBucket in this project).

Similarly, requirement ESYS3211 not only contains the security indicators, but also has increasing comment complexity because of the security audit reports that are generated, the URL's of security reports attached by developers and also the code commits made after fixing the code based on security reports.

Finally, let us consider requirement SYS4542. We notice that the requirement does not contain any security indicators or URL's, the implementation of preview involves the aspect of showing parsed details to stakeholders without high comment complexities.

To understand RQ<sub>2</sub>, we consider some vulnerable packages available in npm and see how they would affect the application when they are downloaded along with their dependencies. In order to understand the effect of derived security implications and explain RQ<sub>2</sub>, let us consider package dependencies in requirement ESYS7372. Package dependencies can usually be a serious security threat as one affected package may be infecting a lot of other packages. While building web applications, it is often very common to use packages or libraries provided by the open source environments. However, the point to be noted is that we need to consider that, any vulnerability introduced by attackers in the packages itself would have an effect not only on that package but any package that inherits it and basically have an impact on the entire application directly or indirectly. One such attack is mirrored below into our case study for more detailed explanation.



```
For module selected
  require(module)
  patch calls to module to our function
Mirror functionality to the attackers server
if(error){
  silence
}
Make original request and handle it.
```

Figure 5.4 Flow of attack

Figure 5.4 models the attack that is introduced into the package. The package which is under attack is require'd. Any calls to this package will be patched to the function being called and attacker gets to mirror the entire call. If the attacker fails by any chance, he may silence the error such that it is not recorded on the user's or developer's log. If the attacker is successful, he can mirror the entire call and then process the actual call, so that the user would never know that he has been attacked.

These types of attacks are extremely dangerous as they do not leave traces. Detection of such attacks are very difficult as they would be embedded inside the packages. When any other package is dependent on the effected package, it is also prone to being vulnerable to security.

We introduce such attacks in our application to test how the package functionality would work.

```

{
  const request = require('request')
  const OriginalRequest = request.Request
  request.Request = (options) => {
    const originalCallback = options.callback
    options.callback = (err, httpResponse, body) => {
      const initiatedRequest = require('http').request ({
        hostname: 'attacker',
        port: 8000,
        method: 'POST'
      })
      initiatedRequest.on('error',() => {})
      initiatedRequest.write(JSON.stringify(body,null,2))
      initiatedRequest.end()

      originalCallback.apply(this,args)
    }
  };
}

```

Figure 5.5: Code snippet of attack introduced in package

Figure 5.5 gives a description of one such code snippet which was introduced into our case study. Here, we attack the request module from an attacker's server running on port 8000. Our web application provides an option where we attach information and upload it in order to send the uploaded files through email. The application provides an option for the users to check if the file attached is the right one; if not they have the option of deleting the file. The above code in Figure 5.5 mirrors the output to the attacker on host 8000 whenever we use a request call even when we are logged into the application after the authentication checks. The code snippet in Figure 5.6 shows how the call is made by the application when anyone clicks on the show files button. When the show files button is clicked get request is initiated and it is supposed to get all the files that have been uploaded.

```
const req = new XMLHttpRequest('GET', 'http://localhost:8080/getallfiles', formdata, {  
  reportProgress: true,  
  responseType: 'text'  
});  
return this.http.request(req);  
}
```

Figure 5.6: Code snippet of call made by web application

On clicking the show files button, the get request is initiated and the request is mirrored to the attacker on server 8000 first, i.e., the attacker gets a copy of the uploaded files first as shown in Figure 5.8. On the other hand, the actual request is also processed normally, and the user gets the list of files as shown in Figure 5.7. The user would not even notice that the data has been shared with the attacker.

Also, notice that any detailing in terms of uploading the file will not sense that an attack is being triggered by the action performed. This attack is a scale down model for the actual attacks that place on the web applications. Attacks like these cause a higher risk to not only security of the application but security of the entire organization in some cases.

As mentioned earlier, the clients of the web application are not generally aware of the attack that is taking place. Even for the developers, tracking down which exact package has a security vulnerability is an extremely difficult and time consuming task.

Select email priority  
High

---

Subject  
Welcome !!

---

B

I

**MyHub:** You will have access to MyHub on your first day of employment at our company. MyHub gives you access to select insurance benefits and to enter in your bank account information so that you receive your paycheck. We receive our paychecks through direct deposit to it's important that you get a bank account soon so that you can receive your paycheck.

**Health Insurance:** Please note, if eligible for health insurance, you are NOT automatically enrolled in health insurance with us if you wish to obtain coverage. You must complete the proper steps to enroll in MyHub. See this link for further information: <http://centerlink.oag.org/content1/33422/>  
You can login using the initial login id and password provided below.  
Login: 2735672  
Password: idAss32jo

The W-4 forms are attached in the welcome document attached.


**Employment Verification Letters:** Please email \_\_\_\_\_ to request employment verification letters for travel, applying for an Ohio license, etc. Please also check out our internal website for international employees at <http://ecenterlink.oag.org/content2/11352/>. This website provides a variety of information including frequently asked questions and information about other visa statuses.

100%

Choose File Welcome

Upload Hide Files

List of Files

 <http://localhost:8080/files/sample.csv> X

<http://localhost:8080/files/Welcome.docx> X

Recipients: (Upload a CSV File containing recipients first name, last name and email address)

Choose File

No file chosen

Preview

Figure 5.7. View of the system after the attack

GET

http://localhost:8000/

Params

Send

Authorization

Headers

Body

Pre-request Script

Tests

TYPE

Inherit auth from parent

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

This request is not inheriting any authorization helper at the moment. Save it in a collection to us parent's authorization helper.

Body

Cookies (1)

Headers (3)

Test Results

Status: 200 OK Time: 43 ms

Pretty

Raw

Preview

JSON

```

1 [
2   "http://localhost:8080/files/sample.csv",
3   "http://localhost:8080/files/Welcome.docx"
4 ]

```

Figure 5.8. View on the attacker's side.

This attack models the effect of the package dependencies entirely and shows us how security of an application can be breached with attacks like this. This attack gives us an idea about the damages that can be done due to package dependency. There are many other similar attacks that can be planned on systems like Cross Site Request Forgery (CRSF) or Cross-Site Scripting (XSS). XSS is a technique using which the attackers can redirect the users to steal their information. Cross Site Request Forgery is a technique in which a forged request is sent to server.

In the similar way when we consider requirement ESYS8781, we notice that the label was non-security when meta data heuristics were used. This was because the factors such as stakeholders and URLs did not play a huge role for this particular requirement. There were code commits and discussions which involved comment complexity but not enough to capture essence of the requirement. On using our semantically enriched heuristics, we notice that this requirement was highly inclined towards lexical affinity of words, i.e., on capturing the entire meaning of the sentence it can be observed that the templates that belong to one user should not be viewable by any other user. The essence of this sentence cannot be captured by just using keyword indicators; it can only be understood by considering the sentence as a whole.

Using such experiments on the web application we could address RQ<sub>3</sub>. In order to understand RQ<sub>3</sub>, we further move on to check our semantic heuristics which have been proposed in Chapter 4 to see if they would produce a different result for the same set of sample requirements that we considered for explaining our meta-data heuristics. Currently, we try to analyze our semantic heuristic manually which is why we have considered all

semantic heuristics mentioned in Chapter 4 and not just derived implications. The idea is to get a picture of semantic heuristics laid over our automated meta-data heuristics to check the behavior of classification of our security requirements.

Table 5.3 gives a summary of all the semantic heuristics that we considered against our sample set of requirements. The most interesting change between Table 5.2 and Table 5.3 is that some of the labels are observed to vary. It is very important to note that, we consider all the non-security labels from Table 5.2, which is obtained by using our classifier built in Chapter 3. Table 5.3 summarizes how  $RQ_2$  is handled using the semantic heuristics. Using the heuristics built in Chapter 4 we can manually check if a particular requirement is actually a security or non-security requirement. However, we mark the final labels using only the effects of derived security implications. These labels are the ones mentioned in Table 5.3. This difference or switch in labels based on meta data heuristics and labels based on semantic heuristics can be seen in Table 5.4,

Table 5.3 is a manual analysis of all the requirements that were identified as non-security by our classifier. In the table we manually analyze all the possible heuristics mentioned in chapter 4. However, in this work, we only concentrate on the effects of derived implications on the classification. It is to be noted that the labels specified in Table 5.3 are purely based on the derived implications heuristic.

Table 5.3: Classification and Analysis using semantic heuristics

Requirement ID	Label	Requirement based ( cw)	Depreciated	Derived
ESYS2124	0	×	✓	×
ESYS1341	0	✓	×	×
ESYS7923	0	×	×	×
ESYS4542	1	×	✓	✓
ESYS8781	0	✓	✓	×
ESYS7372	1	×	✓	✓
ESYS2787	1	✓	×	✓

From Table 5.3 we can see that the combination of words, the package dependencies, and depreciated dependencies have their own effects on the requirements and detected that some of that requirements that were detected as non-security by our classifier are actually security requirements. In the upcoming analysis we look at how the heuristics have influenced the change of certain non-security requirements to security requirements.

From Table 5.4, we can consider the OR operator over the results obtained by the two kinds of heuristics. The OR operator is chosen so that we capture any small trace of security detected by either of the heuristics. Table 5.4 answers RQ<sub>3</sub>, i.e., we can see that requirements that were detected as non-security wrongly could be turned into security requirements using the semantic heuristics. Using the derived implications, the semantically enriched heuristic based labels are chosen from the requirements which have

a criterion of non-security which is why some of the semantically enriched heuristics labels are not available and represented by a ‘-’.

Table 5.4. Comparison between meta data heuristics-based labels and semantic heuristics-based labels

Requirement ID	Meta data heuristics-based labels	Semantically enriched heuristics-based labels
ESYS0015	1	-
ESYS2124	0	0
ESYS1341	0	0
ESYS6132	1	-
ESYS7923	0	0
ESYS3211	1	-
* ESYS4542	0	1
ESYS8781	0	0
* ESYS7372	0	1
* ESYS2787	0	1

From Table 5.4, we can see that “\*” marked requirements have been turned from being non-security to security requirements.

Running a similar analysis on all the requirements we notice that, out of all the non-security requirements extracted from the results of our classifier, 41.8% of the requirements that were labeled as non-security requirements initially could be turned into security requirements based on package dependency considerations.



## CHAPTER 6: CONCLUSION

This work aims at classifying the security and non-security requirements based on certain metrics which are related to meta-data and tries to improve the classification by introducing a new parameter which can be characterized as a semantically enriched heuristic.

The work initially introduced a linear classifier-based approach which shows results where we have achieved an average recall of 92.31% and an average precision of 62.94%. These results are based on the meta-data heuristics (metrics). On analyzing the results, our main aim was to identify all the security requirements correctly. In order to do this, we tried to reduce the number of false negatives. Instead of aiming for excellent recall and precision values, we aimed at increasing the correctly classified security requirements.

In order to do this, we considered only the requirements that were marked as non-security and made efforts to check if they were correctly classified. If so, we considered that the meta-data heuristics was sufficient in classification. If not, we tried to see what aspects of the requirement was causing the security implication. On studying a lot of cases and scenarios as mentioned in Chapter 4, we finally decided to study the in-depth effects of package dependencies. To these subset of non-security requirements, we apply the semantic heuristic we have discussed in detail in this work, package dependencies.

Therefore, this work attempts to pick out and try heuristics which look into deeper aspects of a project than just the issue tracking system. In order to do this, we have considered the very well-known fact and constantly emphasized fact in [40], that security vulnerabilities in packages of front-end libraries have been prevailing for a while now.

Using this package information along with their dependency packages helps us classify projects by not only looking at the top layer aspects that are seen in the issue tracking system, but also helps us consider the system in which the actual implementation and the deployment of requirements take place.

This work contributes to introducing a semantically stronger model than a simple linear classifier. The heuristic introduced helps in adding a human insight to the requirements that were classified as non-security requirements.

In this work, we consider our meta-data heuristics and more semantic heuristics in terms of package dependency graphs and perform an analysis on a web application as a part of our case study. Using this case study, we analyzed the behavior of the requirements built for the implementation of a web application on applying our meta-data heuristics and further moved on to analyzing the dependency graphs of all the packages used in the web application. We used the information provided by the package dependency graphs and paths of project auditing to re-classify our subset of non-security requirements. The analysis showed us that we could turn considerable number of the requirements which were classified as non-security requirements to their right label (security requirements).

## 6.1. Limitations

Apart from the contributions discussed above, there are several limitations in our study:

Firstly, we have tested our linear combinations of features (meta-data) using a linear classifier. We could test the features for other non-linear classifiers and check for improvements in recall and precision. We could train an Artificial Neural Network with our meta-data features and try to improve our results.

Second, while checking the validity of our heuristic inclined towards semantics, we have used only the non-security requirements from our linear classification. As this work aims at finding the security requirements, our main aim is to pick as many security requirements as we can, using our heuristics. So, we try to find heuristics which reduce the misclassification of security requirements. This work shows how the package dependencies can help identify the incorrectly classified security requirements. It would be interesting to see if the behavior/impact of package dependencies on the requirements that were classified as security requirements after the classification using meta-data heuristics.

Finally, the study we have conducted are not projects of different scale and different implementation techniques. While considering the meta-data heuristics for classification of requirements for all these projects, a small amount of change in methodology of calculating the comment complexities has been noted. This could be avoided when requirements for the projects follow common conventions.

## 6.2. Future Work

Based on the previous discussions, we propose the following future work:

1. Considering more complexity metrics.
2. Comparing linear classifier with non-linear classifiers to find the most optimal way to combine all feature values.
3. Using more semantic heuristics in the analysis.
4. Integrating our models to industry common tools, such as JIRA, to provide real-time support.

## REFERENCES

- [1] A. Avizienis, J.-C. Laprie, B. Raandell, C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Transactions on Dependable Secure Computing, Vol. 1, 2004, pp. 11–33.
- [2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”, IEEE Communication Survey and Tutorials, Vol. 17, 2015, pp. 2347–2376.
- [3] J. A. Stankovic, “Research Directions for the Internet of Things”, IEEE Internet of Things, Vol. 1, 2014, pp. 3–9.
- [4] M. Hermann, T. Pentek, B. Otto, “Design Principles for Industrie 4.0 Scenarios”, Proc. the 49th Hawaii International Conference on System Sciences, Koloa, USA, 2016, pp. 3928–3937.
- [5] Y. Liu, Industry 4.0: “A Survey on Technologies, Applications and Open Research Issues”, Elsevier Journal of Industrial Information Integration, Vol. 6, 2017, pp. 1–10.
- [6] G. McGraw, “Software Security: Building Security In”, Addison Wesley Ltd, Boston, MA, 2006.
- [7] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, “Automated Classification of Non-Functional Requirements”, Requirements Engineering, Vol. 12, 2007, pp. 103–120.
- [8] N. Niu, S. Easterbrook, “Analysis of Early Aspects in Requirements Goal Models: A Concept-Driven Approach”, Transactions on Aspect-Oriented Software Development III, Vol. 4620, Springer, Berlin, Heidelberg, 2007, pp. 40–72.

- [9] N. Niu, Y. Yu, B. Gonzalez-Baixauli, N. Ernst, J. C. S. do Prado Leite, J. Mylopoulos, “Aspects across software life cycle: A goal-driven approach”, *Transactions on Aspect-Oriented Software Development III*, Vol. 5560, Springer, Berlin, Heidelberg, 2007, pp. 83–110.
- [10] A. Mahmoud, G. Williams, “Detecting, classifying, and tracing non-functional software requirements”, *Requirements Engineering*, Vol. 21, 2016, pp. 357–381.
- [11] Z. Kurtanović, W. Maalej, “Automatically classifying functional and nonfunctional requirements using supervised machine learning”, *Proc. the 25th IEEE International Requirements Engineering Conference*, Lisbon, Portugal, 2017, pp. 490–495.
- [12] W. Maalej, Z. Kurtanović, H. Nabil, C. Stanik, “On the automatic classification of app reviews”, *Requirements Engineering*, Vol. 21, 2016, pp. 311–331.
- [13] N. Munaiah, A. Meneely, P. K. Murukannaiah, “A domain-independent model for identifying security requirements”, *Proc. the 25th International Requirements Engineering Conference*, Lisbon, Portugal, 2017, pp. 506–511.
- [14] M. Gibiec, A. Czauderna, J. Cleland-Huang, “Towards mining replacement queries for hard-to-retrieve traces”, *Proc. 25th IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium, 2010, pp. 245–254.
- [15] E. Shihab, “Practical software quality prediction”, *Proc. the 30th IEEE International Conference on Software Maintenance and Evolution*, Victoria, BC, Canada, 2014, pp. 639–644.
- [16] S. Mahmood, R. Lai, “A complexity measure for uml component-based system specification”, *Software-Practice and Experience*, Vol. 38, 2008, pp. 117–134.

- [17] Y. Shin, A. Meneely, L. Williams, J. A. Osborne, “Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities”, IEEE Transactions on Software Engineering, Vol. 37, 2011, pp. 772–787.
- [18] Wentao Wang, Kavya Reddy Mahakala, Arushi Gupta, Nesrin Hussein, Yinglin Wang, Artifacts of “A linear classifier based approach for identifying security requirements in open source software development”, Submitted to Journal of Industrial Information Integration. URL <https://goo.gl/qr8Y4W>
- [19] T. J. D. Team, Drools document: 20.8 security management. URL [https://docs.jboss.org/drools/release/7.6.0.Final/drools-docs/html\\_single/index.html#\\_wb.userandgroupmgmt](https://docs.jboss.org/drools/release/7.6.0.Final/drools-docs/html_single/index.html#_wb.userandgroupmgmt)
- [20] J. R. Landis, G. G. Koch, “The measurement of observer agreement of categorical data”, Biometrics, Vol. 33, 1977, pp. 159–174.
- [21] B. Schneier, “Beyond Fear: Software Security: Thinking Sensibly About Security in an Uncertain World”, Springer-Verlag, New York, NY, 2003.
- [22] D. S. Starnes, J. Tabor, D. S. Yates, D. S. Moore, “The Practice of Statistics”, W.H. Freeman and Company, New York, NY, 2014.
- [23] P. C. Vras, E. Villani, A. M. Ambrosio, N. Silva, M. Vieira, H. Madeira, “Errors on space software requirements: A field study and application scenarios”, Proc. the 21st IEEE International Symposium on Software Reliability Engineering, San Jose, CA, USA, 2010, pp. 61–70.
- [24] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. D. McDaneil, A. Sheth, “Taintdroid: An information-flow tracking system for real time privacy monitoring on

smartphones”, Proc. the 9th USENIX Symposium on Operating Systems Design and Implementation, Vancouver, Canada, 2010, pp. 393–407.

[25] W. Klieber, L. Flynn, A. Bhosale, L. Jia, L. Bauer, “Android taint flow analysis for app sets”, Proc. the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis, Edinburgh, UK, 2014, pp. 1–6.

[26] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings”, IEEE Transaction on Software Engineering, Vol. 34, 2008, pp. 485–496. [27] S. Theodoridis, K.

Koutroumbas, Pattern Recognition, 4th Edition, Academic Press, Cambridge, MA, 2008.

[28] A. Murgia, G. Concas, M. Marchesi, R. Tonelli, “A machine learning approach for text categorization of fixing-issue commits on cvs”, Proc. ACM-IEEE International Symposium on Empirical Software Engineering and Measurement 2010, no. 6, Bolzano-Bozen, Italy, 2010.

[29] E. Knauss, S. Houmb, K. Schneider, S. Islam, J. Jürjens, “Supporting requirements engineers in recognizing security issues”, Proc. the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality, Essen, Germany, 2011, pp. 4–18.

[30] A. Dekhtyar, V. Fong, “RE data challenge: Requirements identification with Word2Vec and TensorFlow”, Proc. the 25th IEEE International Requirements Engineering Conference, Lisbon, Portugal, 2017, pp. 484–489.



- [31] E. Wittern, P. Suter, and S. Rajagopalan. 2016. “A Look at the Dynamics of the JavaScript Package Ecosystem”, Int’l Conf. Mining Software Repositories. ACM, 351–361.
- [32] Mitropoulos, D., Karakoidas, V., Louridas, P., Gousios, G., and Spinellis, D. “The Bug Catalog of the Maven Ecosystem”, Mining Software Repositories (New York, New York, USA, 2014), ACM Press, pp. 372–375.
- [33] F. Camilo, A. Meneely, and M. Nagappan. “Do Bugs Foreshadow Vulnerabilities? A Study of the Chromium Project”, Working Conf. Mining Software Repositories, 2015. 269–279.
- [34] J. Cox, E. Bouwers, M. van Eekelen, and J. Visser. “Measuring Dependency Freshness in Software Systems”, Int’l Conf. Software Engineering, 2015 IEEE Press, 109–118.
- [35] F. Massacci, S. Neuhaus, and V. H. Nguyen, “After-life Vulnerabilities: A Study on Firefox Evolution, Its Vulnerabilities, and Fixes”, Proceedings of the Third International Conference on Engineering Secure Software and Systems (ESSoS’11). Springer-Verlag, Berlin, Heidelberg, 2011, 195–208.
- [36] A. Decan, T. Mens, and M. Claes, “On the Topology of Package Dependency Networks — A Comparison of Three Programming Language Ecosystems”, In European Conf. Software Architecture Workshops, ACM, 2016.
- [37] T. Lauinger, A. Chaabane, W. Robertson, C. Wilson, and E. Kirda, “Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In ISOC Network and Distributed System Security Symposium”, 2017.

- [38] J. Williams and A. Dabirsiaghi, “The Unfortunate Reality of Insecure Libraries”. White Paper. Contrast Security, 2017.
- [39] Synk 2017. The State of Open Source Security. <https://synk.io/stateofossecurty/>. (November 2017)
- [40] Alexandre Decan, Tom Mens, Eleni Constantinou, “On the impact of security vulnerabilities in the npm package dependency network”, 2018 ACM/IEEE 15th International Conference on Mining Software Repositories. Gothenburg, Sweden.pp 181-191
- [41] Nan Niu, Juha Savolainen, Zhendong Niu, Mingzhou Jin, and Jing-Ru C. Cheng, “A Systems Approach to Product Line Requirements Reuse”, IEEE System Journal, VOL. 8, No. 3, pp. 827-836
- [42] A.Decan, T.Mens and M.Claes, “An empirical comparison of dependency issues in OSS packaging ecosystems”. Int’l Conf. Software Analysis, Evolution, and Reengineering, 2017, pp. 2-12. <https://doi.org/10.1109/SANER.2017.7884604>
- [43] Alexandre Decan, Tom Mens, and Philippe Grosjean, “An empirical comparison of dependency network evolution in seven software packaging ecosystems”. Empirical Software Engineering (10 Feb 2018). <https://doi.org/10.1007/s10664017-9589-y>
- [44] Felix Schuckert, Basel Katt, Hanno Langweg, “Source Code Patterns of SQL Injection Vulnerabilities”
- [45] Rafael Capilla, Barbara Gallina, Carlos Cedina, “New opportunities for Software Reuse”, 17th International Conference, ICSR 2018

- [46] Ahmed Zerouali, Eleni Constantinou, Tom Mens, Gregorio Robles, and Jesus Gonzalez, “An Empirical Analysis of Technical Lag in npm Package Dependencies”, Barahona
- [47] Nan Niu, Tanmay Bhowmik, Hui Liu and Zhendong Niu, “Traceability-Enabled Refactoring for Managing Just-In-Time Requirements”, 22nd IEEE International Requirements Engineering Conference
- [48] Nan Niu, Amy Koshoffer, Linda Newman, Charu Khatwani, Chatura Samarasinghe, and Juha Savolainen, "Advancing Repeated Research in Requirements Engineering: A Theoretical Replication of Viewpoint Merging", 24th IEEE International Requirements Engineering Conference (RE 2016), Beijing, China, September 12-16, 2016, pp. 186-195
- [49] Nan Niu, Sjaak Brinkkemper, Xavier Franch, Jari Partanen, and Juha Savolainen, “Requirements Engineering and Continuous Deployment”, IEEE Software, Vol 35, Issue 2, 2018, pp. 86-90
- [50] “Package dependency Graph for npm”, TypeFox. <http://npm-dependencies.com/>
- [51] Shahab Salehghaffari, Nan Niu, and Sandeep Reddivari, “An Evidence-Based Software Engineering Evaluation Approach”, International Journal of Advanced Computer Science, Vol 3, No. 1, January 2013, pp.51-55.
- [52] Alejandra Yepez Lopez, Nan Niu, “Soft Systems in Requirements Engineering: A Case Study”, 22<sup>nd</sup> International Conference on Software Engineering and Knowledge Engineering, San Francisco, California, July 2010, pp. 38-41.