Utilizing public repositories to improve the decision process for security defect resolution and information reuse in the development environment

Anja Fonn Salen

Master's thesis in Software Engineering at

Department of Computer science, Electrical engineering and Mathematical sciences,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

May 2021

Western Norway
University of
Applied Sciences

Abstract

Security risks are contained in solutions in software systems that could have been avoided if the design choices were analyzed by using public information security data sources. Public security sources have been shown to contain more relevant and recent information on current technologies than any textbook or research article, and these sources are often used by developers for solving software related problems. However, solutions copied from public discussion forums such as StackOverflow may contain security implications when copied directly into the developers environment. Several different methods to identify security bugs are being implemented, and recent efforts are looking into identifying security bugs from communication artifacts during software development lifecycle as well as using public security information sources to support secure design and development. The primary goal of this thesis is to investigate how to utilize public information sources to reduce security defects in software artifacts through improving the decision process for defect resolution and information reuse in the development environment. We build a data collection tool for collecting data from public information security sources and public discussion forums, construct machine learning models for classifying discussion forum posts and bug reports as security or not-security related, as well as word embedding models for finding matches between public security sources and public discussion forum posts or bug reports. The results of this thesis demonstrate that using public information security sources can provide additional validation layers for defect classification models, as well as provide additional security context for public discussion forum posts. The contributions of this thesis are to provide understanding of how public information security sources can better provide context for bug reports and discussion forums. Additionally, we provide data collection APIs for collecting datasets from these sources, and classification and word embedding models for recommending related security sources for bug reports and public discussion forum posts.

Acknowledgements

First and foremost, I would like to thank my supervisor Tosin Daniel Oyetoyan for all the great guidance and advice whilst working on this thesis. It would not have been possible without his help. I would like to especially thank him for his guidance during running the experiments in the case studies, as well as making it possible to reuse some parts of the experiment framework in his previous paper. I would also like to thank my friends and family for their support throughout this year working on the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software vulnerabilities are increasing security focus as critical and sensitive systems which operate critical infrastructure become increasingly dependent on complex software systems [61]. Consequently, organizations have focused efforts on different methods to identify security bugs before attackers do. Some of those efforts include using threat modelling during design, static analysis tools during implementation [45], and using dynamic analysis tools. Recent efforts are looking into identifying security bugs from communication artifacts during the software development lifecycle [44, 46] as well as using public security information sources to support secure design and development [51]. Security risks are contained in solutions that could have been avoided if the design choices have been analyzed by using public information security data sources. It is therefore important that system architects and developers get easily processable and up-to date security information during system design, development, and bug resolution.

Thus, the primary goal of this thesis is to investigate how to utilize public information sources to reduce security defects in software artifacts through improving the decision process for defect resolution and information reuse in the development environment. The main research question (RQ) is as follows:

• **(RQ)** How can we utilize public information sources to improve security in the software development process?

To effectively address our main RQ, we formulate three sub-research questions as follows:

• **(RQ1)** How can public security information sources be used to improve bug report security classification models?

Existing vulnerability tools suffer from high misclassification rates, with especially high cases of false positives [46]. The hypothesis is that additional information can be leveraged to reduce misclassifications of security bug prediction models.

Furthermore, it has been remarked that modern software engineering is evolving so fast that public forums contain more relevant and recent comments on current

technologies than any textbook or research article [20]. Public forums are now de facto knowledge factory for solving software related problems. Unfortunately, not every solution is security hardened. Solutions copied directly from public discussion forums such as StackOverflow to the production environment have been shown to have security implications [1, 4, 17]. Acar et al. [1] showed that the use of StackOverflow by developers leads to insecurities as developers copy-paste vulnerable solutions. We crafted two additional research questions (RQ2 and RQ3) following the above arguments.

• **(RQ2)** How can public security information sources be used to improve security in online discussion forums?

This thesis will investigate how to extract information from public security sources and use this information to gain additional security context for discussion forum posts.

• **(RQ3)** How can public security information sources be used to improve security in the developer's development environment?

This thesis will investigate how to extract information from public security sources and code sharing platforms, and use this information to build classification models and gain additional security context for vulnerable solutions in the developer's development environment.

This thesis uses both qualitative and quantitative research methods to answer the research questions. First we built a data collection tool for collecting data from public information security sources and discussion forums, we then construct machine learning models - classification models for classifying discussion forum posts or bug reports as security or not-security related, and word embedding models for finding matches between public security sources and public discussion forum posts or bug reports. We then perform case studies on open source bug datasets of software artifacts and datasets containing questions from public discussion forums. To evaluate the models, we compare the classification results to state-of-the-art studies, and perform a qualitative assessment of the suggested matches between (1) public information security sources and bug reports, and (2) public information security sources and public discussion forums.

The results from this thesis show mappings between public information security sources and bug reports, and mappings between public information sources and discussion forum questions. These mappings are shown to give very useful security information to developers during bug resolutions or during software development questions and answers process on discussion forums. In addition, the classification results of the quantitative case study are competitive to state-of-the-art studies. The results demonstrate that using public information security sources can provide additional validation layers for defect classification models, as well as provide additional security context for public discussion forum posts.

This thesis makes the following contributions:

1. Provides understanding of how public information security sources can better provide context for bug reports and discussion forums to reduce security issues in software artifacts.

2. Data collection APIs for collecting datasets from public discussion forums, public information security sources, and code sharing platforms.
3. Classification and word embedding models for recommending related security sources for discussion forum posts and bug reports to give the developer additional security context for security vulnerable solutions.

The thesis is structured as follows: Chapter 2 gives an overview of the theoretical background behind the research this thesis is built upon. Chapter 3 presents the design and implementation of the API developed in this thesis for answering the research questions. Chapter 4 gives a detailed description of the case study conducted in order to answer RQ1. Chapter 5 describes the case study conducted in order to answer RQ2. Chapter 6 gives an overview and description of the idea on how to answer RQ3 in future work. Design choices and challenges faced when developing the API, as well as the case study results are discussed in chapter 7. Related work is presented in chapter 8. Finally, chapter 9 summarizes the results of this thesis, and chapter 10 describes potential work that could be conducted in the future.

Chapter 2

Background

In this chapter, we will give an overview of some of the knowledge the research in this thesis is built upon. It is important to understand the concepts described in this chapter before continuing to the following chapters.

2.1 Public Information Sources

2.1.1 Public Security Sources

In this thesis we argue that providing additional context to questions on discussion forums and bug reports by using security information sources from publicly disclosed cybersecurity vulnerabilities (CVE), common weaknesses (CWE), and attack patterns (CAPEC), can improve solutions containing security vulnerabilities and make stakeholders better evaluate security related reports. In addition, it can support vulnerability models built using source code [70, 52].

The publicly disclosed vulnerabilities (CVE) is a list of records for publicly known cybersecurity vulnerabilities. The mission of the CVE program is to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities. The vulnerabilities are discovered and published by organizations around the world that have partnered with the CVE program [11].

Table 2.1: Examples of CVE records

ID	Description
CVE-2020-0001	In getProcessRecordLocked of ActivityManagerService.java isolated apps are not handled correctly. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation. Product: Android Versions: Android-8.0, Android-8.1, Android-9, and Android-10 Android ID: A-140055304
CVE-2019-0762	A security feature bypass vulnerability exists when Microsoft browsers improperly handle requests of different origins, aka 'Microsoft Browsers Security Feature Bypass Vulnerability'.
CVE-2020-1039	A remote code execution vulnerability exists when the Windows Jet Database Engine improperly handles objects in memory, aka 'Jet Database Engine Remote Code Execution Vulnerability'. This CVE ID is unique from CVE-2020-1074.

The common weaknesses enumeration (CWE) is a community-developed list of software and hardware weakness types that have security ramifications. Weaknesses are flaws, faults, bugs, or other types of errors in software or hardware implementation, code, design, or architecture that could result in vulnerabilities if left unaddressed [12].

Table 2.2: Examples of CWE records

ID	Title	Description
CWE-20	Improper Input Validation	The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	The software constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.
CWE-289	Authentication Bypass by Alternate Name	The software performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor.

The common attack pattern enumeration and classification (CAPEC) is a public list of common attack patterns that aims to give the user understanding of how attackers exploit weaknesses in applications and other cyber-enabled capabilities. Attack patterns are descriptions of the common approaches employed by attackers in order to exploit known weaknesses [10].

Table 2.3: Examples of CAPEC records

ID	Title	Description
CAPEC-39	Manipulating Opaque Client-based Data Tokens	In circumstances where an application holds important data client-side in tokens (cookies, URLs, data files, and so forth) that data can be manipulated. If client or server-side application components reinterpret that data as authentication tokens or data (such as store item pricing or wallet information) then even opaquely manipulating that data may bear fruit for an Attacker. In this pattern an attacker undermines the assumption that client side tokens have been adequately protected from tampering through use of encryption or obfuscation.
CAPEC-130	Excessive Allocation	An adversary causes the target to allocate excessive resources to servicing the attackers' request, thereby reducing the resources available for legitimate services and degrading or denying services. Usually, this attack focuses on memory allocation, but any finite resource on the target could be the attacked, including bandwidth, processing cycles, or other resources. This attack does not attempt to force this allocation through a large number of requests (that would be Resource Depletion through Flooding) but instead uses one or a small number of requests that are carefully formatted to force the target to allocate excessive resources to service this request(s). Often this attack takes advantage of a bug in the target to cause the target to allocate resources vastly beyond what would be needed for a normal request.
CAPEC-217	Exploiting Incorrectly Configured SSL	An adversary takes advantage of incorrectly configured SSL communications that enables access to data intended to be encrypted. The adversary may also use this type of attack to inject commands or other traffic into the encrypted stream to cause compromise of either the client or server.

2.1.2 Public Discussion Forums

Public discussion forums refers to forums for posting questions and receiving answers. An example is StackExchange, which is a network that comprises 173 question-and-answer (Q&A) communities that cover topics in diverse fields [28]. A public discussion forum on StakcExchange allows the users to post questions and answers to questions related to a certain topic or domain. The questions and answers have a voting system, where users are subject to a reputation award process. This voting system helps good questions and answers rise to the top, while simultaneously letting incorrect ones fall to the bottom. Users who provide useful content (questions and answers) are granted more privileges on the site. This reputation system allows the sites on StackExchange to be self-moderating. In addition, the forums allow the user to post code snippets in their questions and answers.

Table 2.4: Examples of posts on StackExchange forums

Forum	ID	Title	Question	Top answer
StackOverflow	1120575	What is the difference between a stack overflow and buffer overflow?	What is the difference between a stack overflow and a buffer overflow in programming?	Stack overflow refers specifically to the case when the execution stack grows beyond the memory that is reserved for it. For example, if you call a function which recursively calls itself without termination, you will cause a stack overflow as each function call creates a new stack frame and the stack will eventually consume more memory than is reserved for it. Buffer overflow refers to any case in which a program writes beyond the end of the memory allocated for any buffer (including on the heap, not just on the stack). For example, if you write past the end of an array allocated from the heap, you've caused a buffer overflow.
AskUbuntu	1332473	Is there any way to use microsoft office in ubuntu without using wine? or any other alternative similar to microsoft office which uses the same format	I am using Ubuntu and I need to do my project work. When I use libre office for the work purpose it stores in odt format and converting from odt to docx format changes the alignment of words and paragraphs in documentation. By reading some blogs and watching videos on Youtube I found wine but,I didn't have license for Microsoft office.so is there any other possible way?	You can't use Microsoft Office in any way without a license ... It is payware. You need an windows emulator like wine to run Microsoft .exe files on any Linux. You can use LibreOffice. It reads and writes MS-Office format files. LibreOffice is free software.
ServerFault	1060208	Is the Host: header required over SSL?	Is the Host: header required over SSL even if the request is not HTTP/1.1? So, if a client connects over SSL, and sends the following request: GET / HTTP/1.0 Should the web server throw a bad request due to the missing Host: header? Should the web server respond with an HTTP/1.0 200 OK response? (the index.html file always exists, so a request to /, would never lead to 403/404)	A HTTP/1.0 request does not need a Host according to the standard, but this header is still usually needed in practice to decide on multi-domain setups which content to serve. But if this header is not present and it is still clear which content to serve, than this content can be served without requiring the header. Note that this has nothing to do with TLS and with the use of SNI.

In our approach, we collect posts from four public discussion forums on StackExchange, namely StackOverflow (SO), AskUbuntu (AU), SoftwareEngineering (SE), and ServerFault (SF).

2.1.3 Code Sharing Platforms

Code sharing platforms allow users to share code and solutions for software related problems. Developers often go to these platforms for solutions to their problems, for example from StackExchange forums as described in 2.1.2 or from sites like ProgramCreek. ProgramCreek is a website consisting of code examples for various programming languages. These code examples are collected from various software projects on GitHub, a code hosting platform for version control and collaboration [27]. In our approach, we extract code from public discussion forums on StackExchange as well as the code sharing platform ProgramCreek.

Example of a code example from ProgramCreek using the java.util.List package in Java:

```
1  @Override
2  protected void encodeStage(final BsonWriter writer, final Unset
       value, final EncoderContext encoderContext) {
3      List<Expression> fields = value.getFields();
4      if (fields.size() == 1) {
5          fields.get(0).encode(getMapper(), writer, encoderContext);
6      } else if(fields.size()> 1) {
7          Codec codec = getCodecRegistry().get(fields.getClass());
8          encoderContext.encodeWithChildContext(codec, writer, fields
               );
9      }
10 }
```

Listing 2.1: Example of code from Programcreek.

2.2 Bug Report Classification

A software bug report is a report describing a bug found by a developer or user of a software system. The bug report should contain information about the bug and steps that allow the developer to reproduce the bug. Security bug reports are bug reports that show some sort of security related vulnerabilities in software products. These are very important to classify correctly, as undiscovered security bug reports can lead to vulnerabilities and weaknesses in the system that can be exploited by a possible attacker.

Bug reports are usually logged in a bug tracking system. A bug tracking system is a software application for tracking reported software bugs in software development systems. These bug tracking system may contain thousands of bug reports where only a few of them are security related [46]. This makes it a challenge to find and categorize unlabelled security bugs, and can easily cause security bugs to go under the radar. Text-based prediction models to help security engineers identify these unlabelled security bugs have been proposed. The proposed models often suffer from high misclassification rates, with especially high rates of false positives (non-security bugs incorrectly labelled as security) [46].

Oyetoyan et al. [44] have shown that using collected security keywords as features to train a text classification model gives promising results. In our approach, we investigate whether using collected features from public security sources can improve the misclassification rates of bug report classification.

2.3 Natural Language Processing

Natural Language Processing (NLP) is defined as the automatic manipulation of natural language by software. Natural language refers to the way we communicate with each other by speech and text [7]. NLP sits at the intersection of computer science, artificial intelligence, and computational linguistics, and is a way for computers to be able to analyze, understand and derive meaning from human language. NLP development in applications can be a challenge, as human speech often is imprecise, ambiguous, and the linguistic structure often depends on complex variables like slang, dialects and social context, whereas programming languages are precise, unambiguous and highly structured [37].

Information retrieval is a problem in NLP, and refers to software programs that deal with the storage, organization, retrieval, and evaluation of information from document repositories, in particular textual information. It is the activity of collecting material, usually of an unstructured nature, that satisfies an information need from within large collections usually stored on computers [53]. An example is when a user enters a search query into a web search engine.

We use two different information retrieval techniques in our approach; Term Frequency-Inverse Document Frequency (see 3.3.1) and Word Embedding (see 3.3.2).

2.4 Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that is based on an idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. In machine learning, the data and result is executed in order to create a program that can be used to predict unknown data. This differs from traditional programming where the data and a set of rules is executed in order to create the result. Machine learning can be broadly categorized into three types - supervised learning, unsupervised learning, and reinforcement learning. In addition, some algorithms use a combination of supervised and unsupervised learning, called semi-supervised learning.

2.4.1 Supervised learning

Supervised learning refers to a type of machine learning approach where the algorithms learn from labeled training data in order to make predictions on future unlabeled data. Supervised learning is divided into two categories; classification problems and regression problems. For regression problems you want to predict real values, whereas for classification problems you want to predict what category the input data belongs to. Bug report prediction falls under the classification category, where we want to predict whether a bug report belongs to the security category or the non-security category.

In this thesis we have used five different supervised learning algorithms; random forest, naive bayes, support vector machines, k-nearest neighbor and logistic regression.

2.4.2 Unsupervised learning

Unsupervised learning refers to a type of machine learning where the algorithms find patterns and relationships in unlabeled data. The goal of unsupervised learning can be to discover hidden patterns in the data or simply to perform feature learning. Typical unsupervised learning tasks include clustering, anomaly detection, neural networks and dimensionality reduction.

2.4.3 Reinforcement learning

Reinforcement learning refers to a type of machine learning where the system interacts with a dynamic environment where it must perform a certain goal (e.g. trying to win in a video game). The system is provided feedback in the form of

rewards and punishments based on its decisions. This way it can learn how to improve its performance.

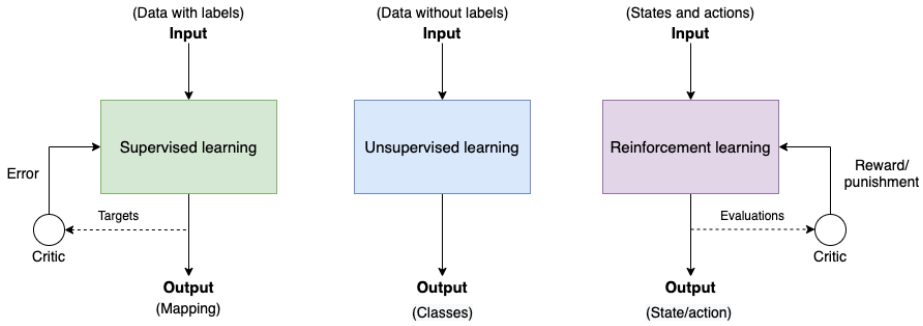

Figure 2.1: Illutration of supervised, unsupervised and reinforcement learning

2.4.4 Overfitting and Underfitting

In machine learning, it is important to create a model that generalizes well on different data. A complex model will often fit the training data better than a simple model, but might perform poorly on new unseen data. This is called overfitting. This is because the model is too closely adapted to the training data to be able to make good predictions on new and unseen data that differs from the training data. On the other hand, a simple model might not be able to make any intelligent predictions based on the underlying relationships in the training data. This is called underfitting. It can often be difficult to find a balance between these two issues.

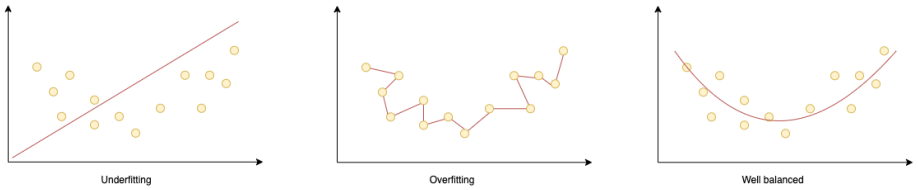

Figure 2.2: Illutration of underfitting, overfitting and well balanced models

2.5 Deep learning and Neural Networks

Deep learning is a sub-field of machine learning that is concerned with algorithms inspired by the structure and function of the brain called artificial neural networks (ANNs) [7]. ANNs try to mimic the operations of a human brain in order to recognize relationships in large amounts of data. Unlike other machine learning techniques, large neural networks tend to increase their performance when we supply them with more and more data. Another benefit of deep learning models is their ability to perform automatic feature extraction from raw data (feature learning) [7].

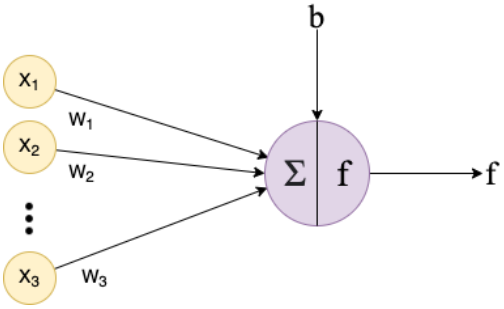

Figure 2.3: Example of neuron

ANNs are based on a collection of connected nodes called neurons. These neurons try to mimic the neurons in a brain, where each connection (called edges) can transmit signals to other neurons. The signal that is transmitted is a real number, and the output of each neuron is usually computed by some non-linear activation function of the sum of its inputs. Each neuron and edge has a weight associated with it that adjusts throughout the learning process. These weights increase or decrease the strength of the transmitted signal.

Figure 2.3 shows an example of a neuron showing the input $X = [x_1, x_2, ..., x_n]$, their corresponding weights $W = [w_1, w_2, ..., w_n]$, a bias (b) and the activation function f applied to the weighted sum. The neuron calculates the dot product between the input signals X and the corresponding weights W as follows:

$$\overline{X} \cdot \overline{W} = b + \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

An additional constant (b) called a bias is added to the weighted sum to give the model more flexibility. The output is then passed through to an activation function f that calculates the final output signal. An activation function is a function added into neural networks that helps the network learn complex patterns in the data. The activation function is what decides what is to be transmitted to the next neuron in the network. Some of the most common activation functions are ReLu, Sigmoid and Tanh. The formulas for these activation functions are as follows:

$$f(z) = max\{z, 0\}(ReLU) \tag{2.2}$$

$$f(z) = \frac{1}{1 + e^{-z}}(Sigmoid) \tag{2.3}$$

$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}(Tanh) \tag{2.4}$$

There are various different models of neural networks, such as feed forward neural networks, recurrent neural networks and convolutional neural networks.

Figure 2.4 shows an example of a feed forward neural network model, where the data is only passed in one direction through all the layers from input to output. Feed forward neural networks tend to be straightforwards and associate the inputs with outputs, and are often used in pattern recognition. This differs from recurrent neural networks, where signals can travel in both directions through loops in the network. These models can get very complicated. Convolutional neural networks (CNNs) contains one or more convolutional layers that create feature maps that records a part of an image, breaks it into rectangular pieces and sends it out for processing. These convolutional layers can be either entirely connected or pooled. The advantage of CNNs are that they can detect important features without human interference.

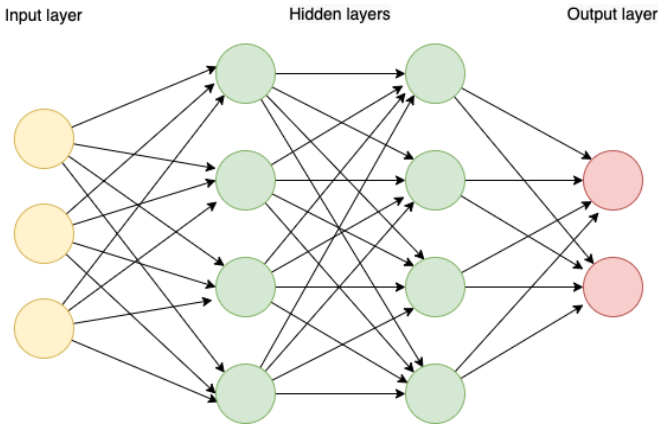

Figure 2.4: Illustration of neural network with two hidden layers

In this thesis we use deep learning in combination with natural lanugage processing (NLP) techniques. Deep learning in the field of NLP promises better performance by models that may require more data but less linguistic expertise to train and operate [7]. Deep learning methods are achieving state-of-the-art results on challenging NLP problems. In our approach, we use a Word Embedding deep learning model (see 3.3.2) for creating a distributed representation of words in a given document. For bug classification we use a word embedding model together with a Convolutional Neural Network (CNN). Word embedding models for representing words are often used together with a CNN for learning how to discriminate documents on classification problems [7]. CNNs are effective at document classification because they are able to pick out features regardless of their position in the document [24].

2.6 Model Evaluation

2.6.1 Bug Report Classification Metrics

The performance evaluation of machine learning models is measured by calculating several metrics. We have used recall, precision, probability of false (pf) alarm, f-score, and g-measure as our performance metrics [54]. Both pf and g-measure are used to compare our work with previous studies [46, 30, 44]. These

metrics are computed from true positive (TP), true negative (TN), false positive (FP), and false negative (FN) where:

TP = Number of security records correctly identified as security records

TN = Number of non-security records correctly identified as non-security records

FP = Number of non-security records incorrectly identified as security records

FN = Number of security records incorrectly identified as non-security records

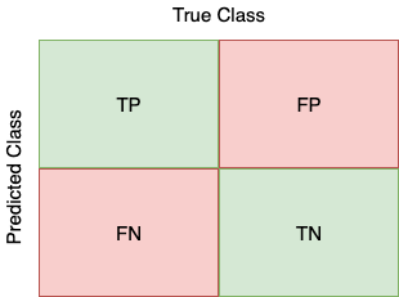

Figure 2.5: Confusion Matrix

Recall shows the sensitivity to classify a bug report as security related or not. It represents the percentage of security related bugs that were correctly classified as security related. A high recall score is desired. The formula for recall is:

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

Precision is the fractions of relevant instances among the retrieved instances. It shows us the percentage of actual security related bugs classified as security related. In practice, having a high recall will decrease the precision, and vice versa. Therefore it is important to look at the specific classification task when choosing which metric is most important. The formula for precision is:

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

The probability of false (pf) alarm is the probability of a non-security related bug being classified as security related. The formula for pf is:

$$\textbf{pf} = \frac{FP}{FP + TN}$$

The F-score combines the precision and recall of the model, and is defined as the harmonic mean of the model's precision and recall. This means that it will have a high value if both precision and recall values are high. The formula for F-score is:

$$\textbf{F-Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

G-measure is the geometric mean of the recall for both the majority and minority classes. The formula for G-measure is:

$$\textbf{G-measure} = \frac{2 * recall * (100 - pf)}{recall + (100 - pf)}$$

2.6.2 Contextualizing Discussion Forum Posts and Bug Reports

When evaluation the results from contextualizing discussion forum posts and bug reports with public security sources, we provide a qualitative assessments of the mappings to external security sources to identify whether they are meaningful or not. When assessing whether a source reasonably matches a discussion or bug report, we use the categories D-Direct, I-Indirect, and N-Not related as our classification scheme. Where "Direct" means the relationship between the two sources are unambiguous, "Indirect" means they can be related (e.g. one provides an example for the other), and "Not related" means we can not infer any relationship between the two.

Chapter 3

Design and Implementation

To answer the research questions require that we first build data collection APIs that interface with public security information sources, public discussion forums, and other code-sharing platforms. Second, it requires that we perform case studies by using the API to collect datasets. In this section, we describe the design and implementation of the data collection API, data collection and pre-processing using the API, information retrieval approaches, case study designs to answer our research questions, and the quality attributes selected for evaluating the API. The source code is available on GitHub from the link specified in appendix A.

3.1 API Development

For this thesis, Java was chosen as the programming language as it was a requirement in the project's description for the thesis. When selecting the framework for the API, we decided on a Apache Maven Java project. Apache Maven is a software project management and comprehension tool that is based on the concept of a project object model (POM) [19]. Maven makes it easy to include necessary dependencies in a project, as well as making it possible to package the project into an executable JAR (Java ARchive) file for distribution.

For the machine learning part of this project, Weka (version 3.8.4) was chosen as the framework for implementing machine learning algorithms and models. Weka is an open source machine learning software that can be accessed through a graphical user interface, standard terminal application, as well as through a Java API [42]. For this project, we chose to use the Java API which allows us to easily build machine learning pipelines and train classifiers.

Deeplearning4j (version 1.0.0.beta6) was used in this project for implementing word embedding (Word2Vec) and neural network models. It is an open-source, distributed deep-learning library written for Java and Scala with wide support for deep learning algorithms [14].

Figure 3.1 gives an overview of the main classes of the API. We have not included the helper classes (such as pre-processing and utility classes) in this diagram as

to not make it too substantial and confusing. The UML diagram shows how the API is put together, and how the different classes relate to each other.

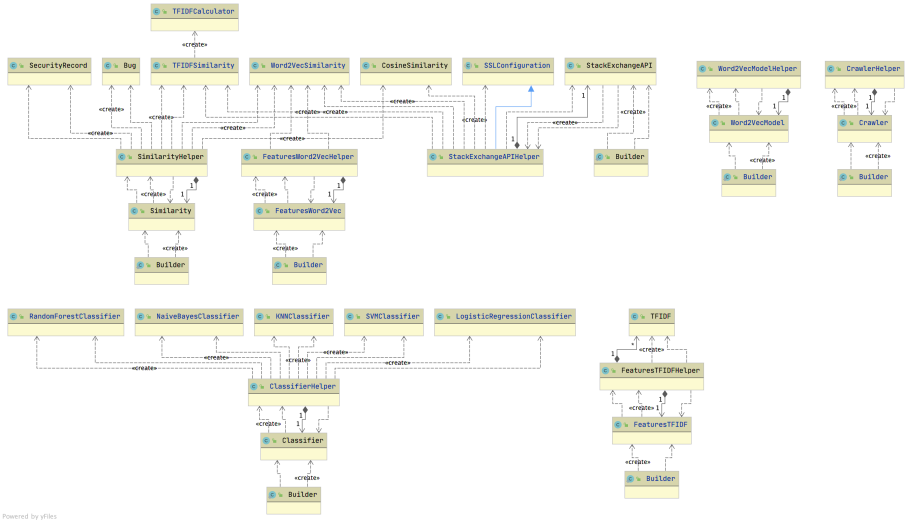

Figure 3.1: UML diagram of the main classes in the API

3.1.1 Interface To Discussion Forums

The API has an interface for sending queries to the StackExchange API, which allows us to query several different public discussion forums. We collect information containing the question title, question description and answers, question ID and date. It is possible to send filtered queries to the StackExchange API, allowing us to send separate queries for collecting security related posts and non-security related posts, as well as adding other tags e.g. Java, IoT, Cloud etc. Additionally, our API has a function for collecting only the code section of a discussion forum post.

3.1.2 Interface To Security Information Sources

The API has an interface for crawling three public security sources; the publicly disclosed vulnerabilities (CVE), common weaknesses enumeration (CWE), and common attack pattern enumeration and classification (CAPEC). The API collects information such as the ID, description, security type, type-of-source, weakness, date, source, reference(s), severity score and severity rating if it is available.

3.1.3 Interface To Code-sharing Platforms

The API has an interface for crawling ProgramCreek for code examples, as well as the option to only extract the code portion of posts from discussion forums on StackExchange. The ProgramCreek crawler in the API supports queries for code examples on the site written in Java, C++ and Scala.

3.1.4 Builder Pattern For User Interaction

The API uses the builder pattern for constructing classes, setting the parameters and running methods. The builder pattern is a design pattern designed to separate the construction of a complex object from its representation so that the same construction process can create different representations [21]. It is a good choice when designing classes with constructors that have more than a handful of parameters. The builder pattern makes it easy for the user to read and understand, which in turn makes it more user friendly.

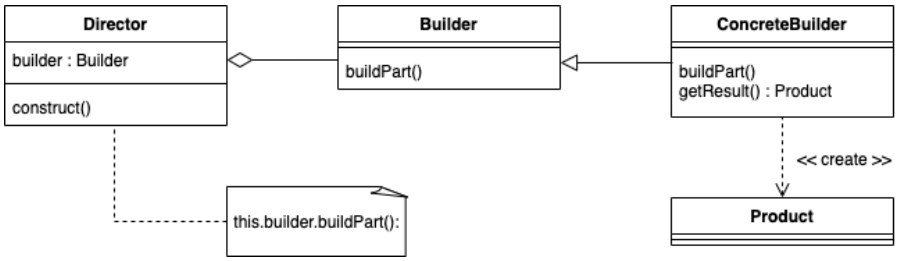

Figure 3.2: UML diagram of Builder Design pattern

Figure 3.2 shows an UML diagram of the builder design pattern. The Product class defines the type of object that is generated by the builder pattern. The Builder is an abstract base class that defines all the steps that must be taken in order to create a product. Each step is abstract as the functionality of the builder is carried out in the concrete subclasses. The ConcreteBuilder may consist of many classes inherited from the Builder, and these classes contain the functionality to create a particular product. The Director is in control of the algorithm for generating the final product object.

Example of builder pattern use in the API:

```
48        StackExchangeAPI sed = new StackExchangeAPI.Builder()
49            .site(StackExchangeAPI.STACKOVERFLOW)
50            .numRecords(1000)
51            .onlyQuestion(true)
52            .security(true)
53                  .tags("java")
54            .pathToStoreResult("./files/")
55            .dataWithoutSimilarity(true)
56            .build();
57
58        sed.run();              // start to collect data
```

Listing 3.1: Using the Builder pattern to query the StackExchange API

3.1.5 Configuration Using Maven

The API is created as a Apache Maven Java Project, which is a software management and comprehension tool that is based on the concept of a Project Object Model (POM) [19]. The POM file contains project information and configuration information for Maven to build the project such as dependencies, build directory, source directory, test source directory, plugins etc. A POM file makes

it easy to include necessary dependencies in a project, as well as update them when a newer version is released.

Code example 3.2 shows how we have included Weka and Deeplearning4j library dependencies so that Maven can compile, build, test and run these libraries.

```
        <dependency>
            <groupId>nz.ac.waikato.cms.weka</groupId>
            <artifactId>weka-stable</artifactId>
            <version>3.8.4</version>
        </dependency>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>deeplearning4j-core</artifactId>
            <version>1.0.0-beta6</version>
        </dependency>
```

Listing 3.2: Examples of dependencies in Maven POM file

3.1.6 Java Properties File

The API uses a Java properties file for storing parameters that should be possible for the user to change as they see fit. The .properties extension is a file extension used in Java related technologies. It is mainly used for storing configurable parameters, where each parameter is stored as a pair of strings. In the API it is used for allowing the user to change the .csv file separator, StackExchange API version, and other parameter variables used for classification like class index and the train/test split ratio.

Code example 3.3 shows examples of parameters that can be set in the Java properties file. Here the user can set the file separator, train size, class index for classification, and the StackExchange API version.

```
# specify csv column separator
SEPARATOR=;
# specify ratio for splitting dataset into train and test
TRAIN_SIZE=0.80
# specify index of class label
CLASS_INDEX=1
# specify Stack Exchange API version
STACKEXCHANGE=2.2
```

Listing 3.3: Examples of parameters in Java properties file

3.2 Data Collection and Preprocessing

Figure 3.3 gives an overview of the dataset collection framework. It shows the steps of the feature extraction methods, how we have built the different filtered and unfiltered datasets used for training a machine learning model later on, as well as how we find the most similar security source mappings for each security related post and bug report.

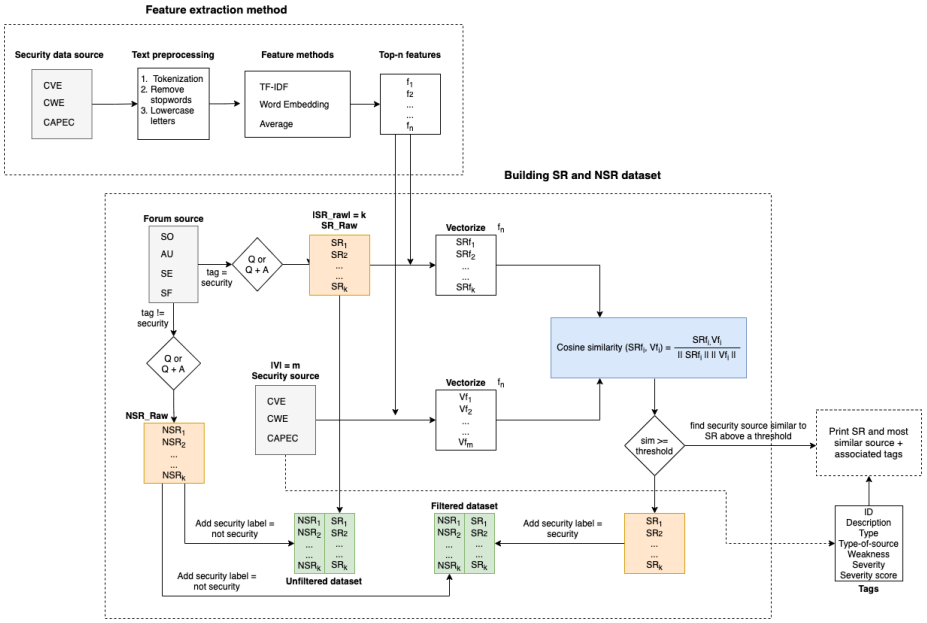

Figure 3.3: Dataset Collection Framework

3.2.1 Public Security Sources

We collected security information from three public security sources; the publicly disclosed vulnerabilities (CVE), common weaknesses enumeration (CWE), and common attack pattern enumeration and classification (CAPEC). As shown in Figure 3.3, we collect the features for filtering datasets from discussion forums (see 3.2.2) and for training machine learning models from these sources. From CVE we have collected 5000 records from 2019 and 2020 due to computational limitations, and from CWE and CAPEC we collected all available records.

3.2.2 Public Discussion Forums

We collected posts from four discussion forums, namely StackOverflow (SO), AskUbuntu (AU), SoftwareEngineering (SE), and ServerFault (SF) where $tag = security$ for security related posts (SRs) and $tag != security$ for non-security related posts (NSRs) as illustrated in Figure 3.3. We have used the StackExchange API v2.2 to collect the data. We separate the SR datasets into two categories - filtered and unfiltered datasets. Filtered datasets are collected by performing a similarity score between a given post and each of the reports in a given security source document (CVE, CWE or CAPEC). We include a discussion if the cosine similarity score to a particular security source is above a certain threshold. For this work, we have used a threshold of 0.4 in order to obtain enough SR records from forums with low security related discussions.

The cosine similarity scores are calculated using three different information retrieval methods; TF-IDF, word embedding, and taking the average of the scores from the two methods. In total, we collected 1000 SR records from each discussion forum. For unfiltered datasets, we simply collected 1000 SR records

without performing similarities to security sources. We have used 1000 records for NSRs. The collected posts were then preprocessed by removing stopwords, numbers and any non-alphabethic characters before being used to train a model for bug report classification.

3.2.3 Security bug datasets

We have used security bug report datasets from five projects used in three previously published studies [46, 30, 44] as the ground truth for validating our classification models. These projects are: Derby, Camel, Wicket, and Ambari from the Ohira et al. [43] dataset. The fifth project is Chromium from Peters et al. [46]. We chose these datasets because they are publicly available and have been previously used in the research community for validating security bug report classification models. Table 3.1 provides the properties of the five projects while Table 3.2 lists the properties of the actual validation datasets as used in previous studies.

Table 3.1: Validation Project Properties

Project	Domain	Start Date	End Date	#BRs	#SBRs	SBR(%)
Ambari	Hadoop management web UI backed by its RESTful APIs	Sep 26 2011	Aug 8 2014	1000	29	2.9
Wicket	Component-based web application framework for Java programming	Oct 20 2006	Nov 9 2014	1000	10	1.0
Camel	A rule-based routing and mediation engine	Jul 8 2007	Sep 18 2013	1000	32	3.2
Derby	A relational database management system	Sep 28 2004	Sep 17 2014	1000	88	8.8
Chromium	Web browser	Aug 30 2008	Jun 11 2010	41940	192	0.5

Table 3.2: Validation dataset

Project	#BRs	#SBRs	SBR(%)
Ambari	500	7	1.4
Wicket	500	6	1.2
Camel	500	18	3.6
Derby	500	42	8.4
Chromium	20970	115	0.5

3.3 Information Retrieval and Feature Extraction Approaches

Information retrieval refers to finding material of an unstructured nature that satisfies an information need from within large collections [53]. Feature extraction is an approach that provides solutions to information retrieval problems. Feature extraction refers to extracting and producing feature representations that are relevant to the NLP task we are trying to accomplish and the type of

model we are planning to use. In our proposed approach, we have selected two feature extraction approaches; Term Frequency-Inverse Document Frequency (TF-IDF) [53] and the word embedding model Word2Vec [39].

3.3.1 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency, shortened to TF-IDF, is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. It is often used as a weight in information retrieval and text mining.

TF-IDF combines the term frequency (TF) metric and the inverse document frequency (IDF) metric. Term frequency for a term in a document is computed as the fraction of the number of times the term occurs in the document to the total words in the document. IDF on the other hand finds the log of the fraction of the total number of documents to the number of documents where a term appears. The intuition behind IDF is that terms that are frequent in all documents may not discriminate very well and will thus be penalized with low IDF. Conversely, terms that occur in a few documents may be more interesting for the documents where they appear and will thus be weighted with a higher IDF.

The TF-IDF for term t, in document d, in an entire documents D corpus can be computed as:

$$\textbf{TF-IDF(t,d)} = \frac{count(t,d)}{\mid d \mid}.log\frac{\mid D \mid}{|\{d \in D : t \in d\}|} \tag{3.1}$$

By computing the TF-IDF for the entire corpus, we retrieve the top-n features with the highest TF-IDF.

The API also includes methods for using normalized term frequency and boolean term frequency:

$$\textbf{Normalized Term Frequency} = 0.5 + \frac{0.5 * tf_{t,d}}{max_t(tf_{t,d})} \tag{3.2}$$

Where tf(t,d) represents the number of times t appears in a document d and max(t) represents the frequency of the term with the maximum occurrence in a document d. We use 0.5 as the smoothening term to avoid a large swing in the ntf(t,d) from a small change in tf(t,d).

Boolean term frequency prints 1 if the term is present document, and 0 if the term is not present.

3.3.2 Word Embedding

Word embedding is a way of representing a document vocabulary that captures the context of a word in a document, its relation to other words, and its semantic similarity. The basic idea is that if two words share similar contexts, then they will be associated with vectors that are close to each other in the vector space. It is based on distributional semantic models (DSMs), which are based on the

assumption that words appearing in a similar context tend to have a similar meaning [26]. These models represent each word by a d-dimensional vector of real numbers, and words appearing in a similar context have similar vector representations. Traditional DSMs tend to use count-based models, but models based on deep neural networks have recently been proposed [40], [39]. These models use neural networks to learn from the context of the corpus, and create low-dimensional word vector representations (word embedding).

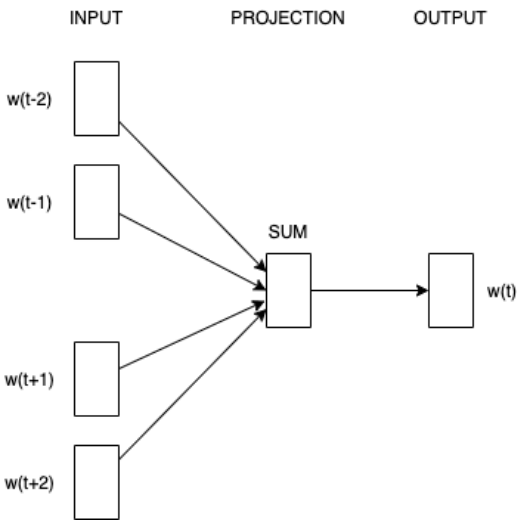

Figure 3.4: CBOW Model Architecture

There are two main word embedding models used; Continuous Bag-of-Words Model (CBOW) and Continuous Skip-gram Model [39], [40]. CBOW predicts a word based on the surrounding context consisting of a fixed window size. As the order of the words in the context is not important, this is called a bag-of-words model. The Continuous Skip-gram model uses the current word to predict the surrounding context words. Figure 3.4 and 3.5 depicts the architecture of CBOW and Skip-gram models, where w(t) represents the current word and w(t $\pm$ 1,2,...,n) represents the surrounding word of the context w(t).

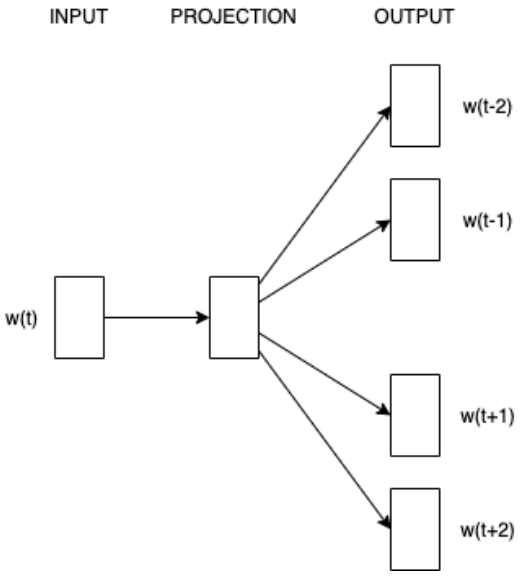

Figure 3.5: Skip-Gram Model Architecture

In our API, we have used the word embedding technique proposed in [39] aka Word2Vec, with an implementation based on the Skip-gram model. The objective function of the skip-gram model is to maximize the negative log likelihood of the surrounding context words (w_{i+k}), with a fixed window size m, conditioned on the center word (w_i) over n vocabulary words.

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{-m\leq k\leq m, k\neq 0}^{n} log\, p(w_{i+k}|w_i) \tag{3.3}$$

The Word2Vec model after training will thus contain a dictionary of words, where each word is associated with a vector representation. We trained a Word2Vec model for each of CVE, CWE, and CAPEC using a window size of 5, and a dimension of 100. By using the trained Word2Vec model, a document d (discussion question or security source record) with $w_1, w_2, ..., w_n$ words can use the mean vector representations of the words. If each word, w_i has u_{w_i} vector embedding, the mean vector embedding for d is computed as:

$$u_d = \frac{1}{n}\sum_{i=1}^{n} u_{w_i} \tag{3.4}$$

We use the mean of the vectors of a discussion or security source record when collecting filtered datasets, and we only select n words as features from the vocabulary list for building a machine learning model.

3.4 Similarity Measures

The main component of the API is to compute similarity between a security source document and a given public information source post or bug report. This

can then later be used to create datasets for building models for classification of bug reports, as well as for tagging public information sources. In order to do this, we use TF-IDF, Word2Vec or a combination of both to transform a given document into vectors and compute the cosine similarity between these vectors. A bigger cosine similarity value indicates that the corresponding two documents are more similar.

Cosine similarity is a metric used to measure how similar two documents are. It measures the cosine of the angle between two vectors projected in a multi-dimensional space. It is calculated as follows:

$$\textbf{Cosine Similarity(A, B)} = \frac{A.B}{\|A\| \times \|B\|} \tag{3.5}$$

The bigger the cosine value the similar the two documents are to each other. We compare two documents that have been vectorized using TFIDF or Word2Vec model by computing their cosine similarity.

3.4.1 TF-IDF approach

Given a security source, we first extract the top n terms from the descriptions in the documents using either TF-IDF or Word2Vec. It is then possible to create TF-IDF word vectors for a security source document and a public information source document or bug report using either the TF-IDF top n terms or the Word2Vec top n terms. We can then calculate the cosine similarity between the two documents to generate a similarity score.

3.4.2 Word Embedding Approach

In this approach, we use a Word2Vec model based on a given security source to create word embedding vectors for a security source document and a public information source document or bug report. We can then calculate the cosine similarity between the two documents to generate a similarity score.

3.5 Case study design to answer RQ1 and RQ2

3.5.1 Bug classification and contextualizing with Word2Vec (RQ1)

In this case study we explore RQ1 by investigating whether using public security sources and public discussion forums for building datasets and training a machine learning model will improve misclassificaton rates in bug classification models. In addition, we investigate whether we can use security information sources from publicly disclosed cybersecurity vulnerabilities (CVE), common weaknesses (CWE), and attack patterns (CAPEC) to further provide additional context to the bug report predicted by the bug report classification models. We are especially interested in whether non-security bug reports predicted as security can be exposed as security vulnerable with the additional context. The case study is described in detail in chapter 4.

3.5.2 Public forum classification and contextualizing with Word2Vec (RQ2)

In this case study, we aim to answer RQ2 by investigating whether providing additional context to questions on discussion forums and bug reports by using security information sources from publicly disclosed cybersecurity vulnerabilities (CVE), common weaknesses (CWE), and attack patterns (CAPEC) can improve the security in online discussion forums. Our hypothesis is that security information sources can provide an additional validation layer of security classification for public information sources. The case study is described in detail in chapter 5.

3.6 Quality Evaluation of the API

Software quality is the degree to which software possesses a desired combination of attributes [IEEE 1061]. Developers of critical systems are responsible for identifying the requirements of the application, developing software that implements the requirements, and for allocating appropriate resources [5]. Software quality attributes include attributes like correctness, reliability, adequacy, learnability, robustness, maintainability, readability, extensibility, testability, efficiency, portability and usability. High scores in these attributes enables developers to guarantee that a software application will meet the specifications of the client. In the development of the API, we have focused on three main software quality attributes; usability, maintainability and extensibility. In addition, we have performed a static code analysis to check the internal quality attributes and dependencies in the API.

3.6.1 Usability

Usability refers to how well a specific user in a context can use a product/design to achieve a defined goal effectively, efficiently and satisfactory [18]. According to The Interaction Design Foundation [18], the product/design should contain these elements in order to accommodate users' needs and context:

- **Effectiveness** — It should support users in completing actions accurately.
- **Efficiency** — Users should be able to perform tasks quickly through the easiest process.
- **Engagement** — Users find it pleasant to use and appropriate for its industry/topic.
- **Error Tolerance** — It supports a range of user actions and only shows an error in genuine erroneous situations. You achieve this by finding out the number, type and severity of common errors users make, as well as how easily users can recover from those errors.
- **Ease of Learning** — New users can accomplish goals easily and even more easily on future visits.

In our approach, we have chosen to use the Builder Design Pattern described in 3.1.4 to build in the usability quality attribute. The builder pattern makes

it easy for the user to read and understand, which in turn makes it more user friendly. It has a simple setup, and makes it easy and quick for the user to complete their desired task correctly, as well as allowing the user to change a variety of parameters to fit their needs.

In addition, we use a Java Properties File as described in 3.1.6 where the user has the possibility to configure parameters that are used throughout the API, like the .csv file separator, StackExchange API version, and other parameter variables used for classification like class index and train/test split ratio.

3.6.2 Maintainability

Maintainability refers to the suitability for debugging and for modification and extension of functionality [3]. According to Quality Assurance [3], the maintainability of a software system depends on these elements:

- **Readability** - The form of representation, programming style, consistency, readability of the programming language(s), structuredness of the system, quality of the documentation and tools available for inspection.
- **Extensibility** - Allowing required modifications at the appropriate locations to be made without undesirable side effects.
- **Testability** - Suitability for allowing the programmer to follow program execution and for debugging.

In order to support this quality attribute, we have used Maven POM as described in 3.1.5. The POM file contains project information and configuration information for Maven to build the project, and allows the user to easily update dependencies and plugins etc. when a newer version is released. It is also easy for the user to add new dependencies if it is desired. In addition, using the Builder design pattern described in 3.1.4 makes the code very maintainable as it is very easy to change the implementation of an object, add or remove parameters, and add, remove or change implemented methods.

3.6.3 Extensibility

Extensibility should allow for required modifications at the appropriate locations to be made without undesirable side effects [3]. According to Quality Assurance [3], the extensibility of a software system depends on these elements:

- Structuredness (modularity) of the software system.
- Possibilities that the implementation language provides for this purpose.
- Readability (to find the appropriate location) of the code.
- Availability of comprehensible program documentation.

Extensibility is an element of maintainability, and thus require some of the same measures. As described in 3.6.2, we have used Maven POM for including dependencies and plugins that can easily be updated and extended. In addition, the Builder Design Pattern described in 3.1.4 makes the code very readable and extendable because of the clear separation of responsibility between the classes. This way, the API can easily be extended, for instance by adding new

classification algorithms, new feature extraction methods, or even new crawlers for public information sources or code sharing platforms.

3.6.4 Static Code Analysis

Static code analysis is a way of debugging and examining code before the program is run. It is typically done by analyzing the code against a set of rules. We have used the static code analysis tool CodeMR, which gives a quick and simple insight into the software quality of the code [38]. CodeMR gives then user several graphs that explain how the code scores in the different quality attributes. Coupling, Complexity, Cohesion and Size are the fundamental internal quality attributes of code [38]. Coupling is the degree of interdependence between classes, measuring how closely connected two classes are. Complexity is how complex the code is, where if it has high complexity, the code is difficult to understand. Cohesion measures how well the methods in a class are related to each other. High cohesion (low lack of cohesion) is usually preferable. Size refers to the number of lines and methods of a class. Bigger classes can be harder to maintain, and might do too much work that could possibly be split up into several classes.

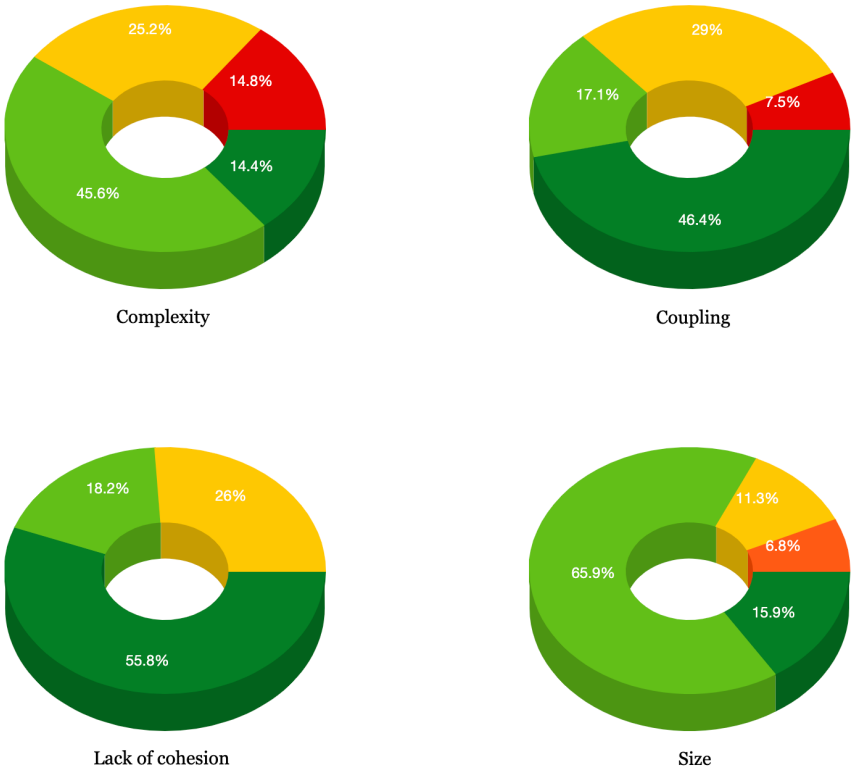

Figure 3.6: CodeMR graph results

Figure 3.6 shows the percentage of classes in the API that fall under a certain

category of each of the four quality attributes. In the graphs, dark green means a low score, green means low-medium, yellow means medium-high, orange means high, and red means a very high score. Only the complexity and coupling metrics have classes that score very high (red), where three classes score very high in complexity and one class scores very high in coupling. One class scores high (orange) in size. Apart from that, most classes score low (dark green) or low-medium (green) in all metrics, and a few score medium-high (yellow). Only one class has both high coupling and high complexity. According to the CodeMR results, the rest of the classes have fair quality attributes.

Figure 3.7 shows how the different packages in the API depend on each other. A dependency is another package that a given package depends on in order to work. Most of the dependencies are to the similarity package, which contains all the classes that compute similarity. The only exception is the machine learning package, which depends on the classifiers package containing all the classes for the different classification algorithms. The other dependencies are to the utility package which contains most of the helper classes, for example classes containing methods for removing symbols from text or pre-processing methods for the classification algorithms.

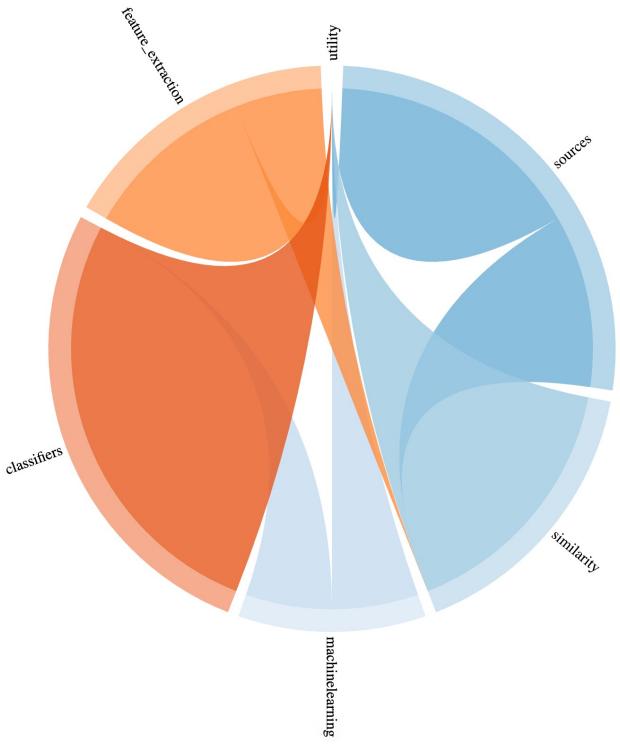

Figure 3.7: CodeMR dependency graph

Chapter 4

Case study 1: Contextualizing Bug Reports Using Public Security Data Sources

In this case study we explore RQ1 by investigating whether using public security sources and public discussion forums for building datasets and training a machine learning model will improve misclassificaton rates in bug classification models. In addition, we investigate whether we can use security information sources from publicly disclosed cybersecurity vulnerabilities (CVE), common weaknesses (CWE), and attack patterns (CAPEC) to further provide additional context to the bug report predicted by the bug report classification models. We are especially interested in whether non-security bug reports predicted as security can be exposed as security vulnerable with the additional context. To address the research question, we build classification models from public discussion forum datasets and public security datasets using the TF-IDF and Word2Vec feature extraction methods shown in Figure 3.3. These classification models are then validated using open source bug reports. We then use the best scoring classification model to make predictions on an open source bug report, and use a Word2Vec model to provide mappings between the predicted bug reports and security sources in order to provide additional context.

To answer RQ1, we investigate two subquestions:

- **RQ1.1:** How can public security information sources be used to improve performance of bug report classification models?
- **RQ1.2:** How can public security information sources be used to improve security in bug report classification models?

4.1 Experiment setup and modelling approach

We build classification models from public discussion forum datasets and public security datasets using the TF-IDF and Word2Vec feature extraction methods shown in Figure 3.3. Table 4.1 presents the different combinations of parameters for building our classification models. In total, we constructed 6480 models. We have used six text classification algorithms that have been commonly used in the research environment [46] - Random Forest, Naive Bayes, Support Vector Machine, K-Nearest Neighbor, Logistic Regression, and Multilayer Perceptron Layer (CNN). Both the Weka (version 3.8.4) [42] and DeepLearning4J (version 1.0.0.beta6) [14] libraries are used for our experiments. We have used default parameters in the algorithms.

As shown in Figure 4.1 we perform five experiments for each combination of parameters and compute the mean, maximum, minimum, and standard deviation for the performance metrics. In each experiment, we train a model using 5-fold cross-validation where the training dataset is split into 4 folds (80%) for training and 1 fold (20%) for testing in each round. The model is then validated using each of the security bug validation datasets.

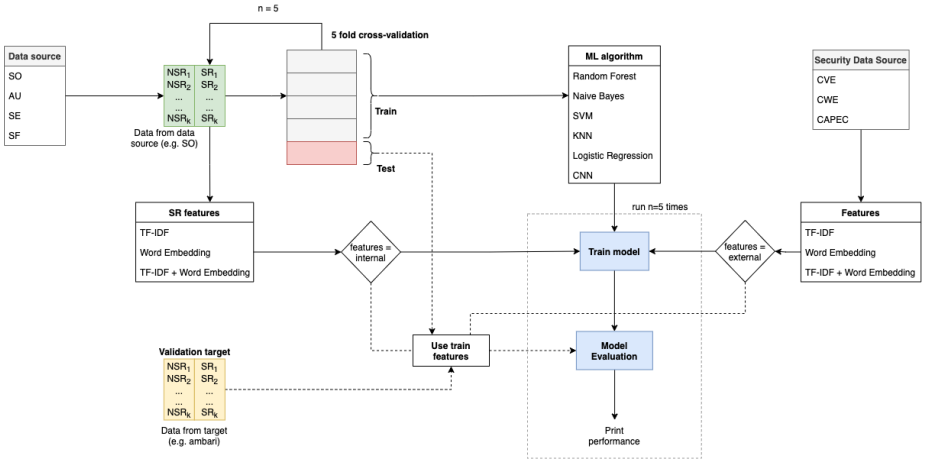

Figure 4.1: Experiment modeling framework

Table 4.1: Experiment detail

Models	Total
3 security sources (CVE, CWE, CAPEC) 4 discussion forums (SO, SE, SF, AU) 2 Content categories (Q, Q+A) 5 validation dataset (Ambari,Wicket,Camel,Derby,Chromium) 6 Machine Learning Algorithms 3 Feature types (TFIDF, Word2Vec, Both.) 3 Feature dimensions (100, 200, 300)	6480

4.1.1 Research questions and analysis

RQ1.1: How can public security information sources be used to improve performance of bug report classification models?

We approach RQ1.1 in 2 ways:

1. **Comparison to state of the art studies**: We first compare the results of our model to state-of-the-arts studies [46, 30, 44]. We have compared our results to those reported for transfer project prediction since we have used public discussion forums to construct the training dataset for our classification model. We then use the same test dataset as used in all these studies to validate our model. A comparable result to the state-of-the-art studies will be a proof of the usefulness of our approach.

2. **Hypothesis**: We hypothesize that using public security sources to filter training datasets for constructing classification models can produce better classification models. Our null hypothesis (H_0) is thus: Models based on unfiltered datasets by security sources outperform models based on filtered datasets.

To test the hypothesis, we collect the top-50 observations ranked by g-measure for each of the models that use filtered datasets and unfiltered datasets. The data is unpaired, thus we use Wilcoxon signed ranked test [15] (a non-parametric test) at 95% confidence level to compare the mean of both groups for three metrics - recall, f-score, and g-measure. In addition, we perform an effect size check on our results. As noted in Kampenes et al. [31], effect size quantifies the size of the difference between two groups and allows us to judge whether the conclusions drawn from our hypotheses testing are meaningful or not. It might be possible that the effect size is negligible even when the statistical test is significant and vice versa. We apply the Hedges, g standardized effect size measure calculated as: $Hedges, g = \frac{\bar{X}_1 - \bar{X}_2}{S_p}$. Where $\bar{X}_1$ and $\bar{X}_2$ represent the sample means for the classification measures (recall, f-score, and g-measure) and S_p represents the pooled standard deviation computed from the standard deviations of s_1 and s_2 of the two groups. We use the results reported in Software Engineering empirical studies categorized under Table 9 as the basis for comparing our effect sizes. The size category for 284 estimated values for Hedges, g is given as: Small: 0.00-0.376, Medium: 0.378-1.000 and Large: 1.002-3.40. We have used standard statistical packages in R [49].

RQ1.2: How can public security information sources be used to improve security in bug report classification models?

To answer RQ 1.2, we output the classification results for the Chromium dataset, and manually validate the true positives and false positives predicted by the model. We use the best classification model on the Chromium dataset shown in Table 4.2 for outputting the prediction results. We use a Word2Vec model that we have trained on each of the security sources. For each record, we use the mean vector embedding from the Word2Vec model described in 3.3.2 and then perform a cosine similarity to the mean vector embedding of each record from the security source. Using a cosine of 0.5 as our threshold for the true positives and 0.4 for the false positives, we collect the top five most similar security sources to

a particular bug report. We select the first 100 bug reports predicted to be true positives from Chromium. We then provide a qualitative assessments of these mappings to external security sources to identify whether they are meaningful or not. When assessing whether a source reasonably matches a bug report, we use the method described in 2.6.2. We have evaluated the 100 true positive records, where a record has one to five suggested matches. We record a one for a category when at least one of the five matches is marked with this category. For the false positive records, about 100 bug reports were evaluated.

4.2 Results & Discussion

Table 4.2: Comparison to state-of-the-art studies

Target	Source	Paper (Model)	Cat	Ratio	Learner	TN	TP	FN	FP	pd	pf	prec	f-score	g-measure
Chromium	Ambari	Peters et al. (clnifarsecsq)	-	-	MLP	19,817	56	59	1,038	48.7	5.0	5.1	9.3	63.9
	Derby	Jiang et al. (rs-selector)	-	1 : 1	MLP	15,346	76	39	5509	66.1	26.4	1.4	2.7	69.6
	Camel	Oyetoyan et al.(fsec-ext+)	TCAI	2.0	LR	16,737	84	31	4109	73.0	19.7	2.0	3.9	76.5
	AU	$CWE-Q-AVG-300-Both(Ext)$	-	-	RF	17,626	72	43	3229	62.6	15.5	2.2	4.2	72.0
Wicket	Camel	Peters et al. (train)	-	-	NB	437	3	3	57	50.0	11.5	5.0	9.1	63.9
	Ambari	Jiang et al. (rs-selector)	-	2 : 1	MLP	396	6	0	98	100.0	19.8	5.8	10.9	88.9
	Camel	Oyetoyan et al.(fsec-ext+)	TCAI	2.0	LR	412	6	0	82	100.0	16.6	6.8	12.8	90.9
	SO	$CAPEC-QA-TFIDF-300-Both(Int)$	-	-	RF	396	6	0	98	100.0	19.8	5.8	10.9	88.9
Ambari	Chromium	Peters et al. (farsecsq)	-	-	MLP	474	3	4	19	42.9	3.9	13.6	20.7	59.3
	Chromium	Jiang et al. (rs-selector)	-	3 : 1	LR	417	7	0	76	100	15.4	8.4	15.6	91.6
	Derby	Oyetoyan et al.(fsec-tfidf+)	-	0.0	SVM	415	6	1	78	85.7	15.8	7.1	13.2	84.9
	SE	$CWE-Q-TFIDF-200-Both(Int)$	-	-	KNN	414	6	1	79	85.7	16.0	7.1	13.0	84.8
Camel	Derby	Peters et al. (farsectwo)	-	-	NB	371	8	10	110	44.4	22.9	6.8	11.8	56.4
	Derby	Jiang et al. (rs-selector)	-	9 : 1	LR	425	13	5	57	72.2	11.8	18.6	29.6	79.4
	Derby	Oyetoyan et al.(fsec-ext+)	TCAI	1.0	LR	359	13	5	123	72.2	25.5	9.6	16.9	73.3
	AU	$CAPEC+Q+Word2Vec-200-Both(Ext)$	-	-	KNN	371	14	4	111	77.8	23.0	11.2	19.6	77.4
Derby	Chromium	Peters et al. (clnifarsecsq)	-	-	NB	372	19	23	86	45.2	18.8	18.1	25.9	58.1
	Ambari	Jiang et al. (ms-selector)	-	2 : 1	MLP	295	30	12	164	71.4	35.7	15.5	25.4	67.7
	Wicket	Oyetoyan et al.(fsec-ext+)	C	2.0	LR	419	27	15	39	64.3	8.5	40.9	50.0	75.5
	SE	$CWE-QA-Word2Vec-300-TFIDF(Ext)$	-	-	NB	360	30	12	98	71.4	21.4	23.4	35.3	74.8

4.2.1 Comparison to state-of-the-art studies

We compare the best results of the models built with discussion forums to other models reported in the literature. For Chromium, the model stands in the second place with 72% gmeasure, 4.2% fscore, and 62.6% recall. For Wicket, it ranks second with a g-measure of 88.9%, 10.9% fscore, and 100% recall. The model ranked third for Ambari with 84.8% gmeasure, 13% fscore and 85.7% recall. For Camel, the model ranked second with 77.4% gmeasure, 19.6% fscore, and best recall of 77.8%. Lastly, for Derby, the model also ranked second with 74.8% gmeasure, 35.3% fscore, and 71.4% fscore. Clearly, the results are competitive to existing results.

We can infer that the public discussion forum contains textual reports that can be semantically similar and useful to build security bug report classification models.

Table 4.3: Summary of results of filtered dataset with the best average g-measure

								pd		pf		prec		fscore		g-measure	
Target	Forum	Content	Feature.Method	Feature.source	Feature.dim	Feature.Learner	Learner	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev
Chromium	SO	Q	Word2Vec	CVE	200	Both(Ext)	LR	52.1	1.0	8.5	2.6	1.9	0.5	3.7	0.9	66.4	0.4
	SF	Q	TFIDF	CWE	300	Both(Ext)	RF	53.9	1.3	11.5	1.6	1.9	0.2	3.7	0.4	66.6	0.6
	SO	Q	AVG	CAPEC	200	Word2Vec(Ext)	LR	52.2	0.9	8.9	1.5	2.3	0.4	4.4	0.6	66.4	0.6
Wicket	AU	Q	Word2Vec	CWE	300	TFIDF(Int)	RF	83.3	0.0	9.3	2.2	6.0	1.3	11.2	2.1	83.8	1.1
	AU	QA	TFIDF	CAPEC	300	Word2Vec(Int)	NB	66.7	0.0	5.9	1.2	7.8	1.4	14.0	2.1	76.8	0.4
	SO	Q	AVG	CWE	100	Both(Ext)	NB	66.7	0.0	13.6	2.0	3.9	0.6	7.5	1.0	72.9	0.8
Ambari	AU	QA	Word2Vec	CWE	300	TFIDF(Ext)	RF	71.4	0.0	10.3	0.9	7.1	0.6	12.9	0.9	78.4	0.4
	SO	QA	TFIDF	CAPEC	100	TFIDF(Ext)	RF	71.4	0.0	10.7	0.9	7.1	0.5	12.9	0.9	78.4	0.4
	SE	Q	AVG	CWE	100	Word2Vec(Ext)	RF	71.4	0.0	8.1	0.7	8.9	0.7	15.9	1.1	79.5	0.3
Camel	AU	Q	Word2Vec	CAPEC	200	Both(Ext)	RF	72.2	2.2	23.0	1.4	9.1	0.8	16.1	1.3	72.6	1.8
	AU	Q	TFIDF	CAPEC	200	TFIDF(Ext)	RF	72.2	2.2	27.6	2.4	7.3	0.5	13.3	0.8	68.8	1.2
	AU	Q	AVG	CAPEC	200	Word2Vec(Ext)	RF	66.7	2.2	23.2	1.4	8.6	0.4	15.2	0.6	69.9	1.0
Derby	AU	Q	Word2Vec	CWE	200	TFIDF(Ext)	SVM	59.5	1.8	23.6	0.7	18.2	0.6	28.1	0.9	66.9	1.1
	AU	Q	TFIDF	CWE	200	TFIDF(Ext)	NB	66.7	3.6	25.1	5.2	14.1	1.9	23.5	2.3	65.2	1.8
	AU	Q	AVG	CWE	300	TFIDF(Ext)	SVM	52.4	4.6	17.2	6.2	15.5	2.7	24.8	2.6	64.2	0.8

Table 4.4: Summary of results of unfiltered dataset with the best average g-measure

							pd		pf		prec		fscore		g-measure	
Target	Forum	Content	Feature.source	Feature.dim	Feature-Learner	Learner	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev
Chromium	SE	Q	CVE	200	Word2Vec(Ext)	LR	54, 8	0, 9	13, 2	1, 3	1, 9	0, 2	3, 7	0, 3	67, 1	0, 5
	SF	Q	CWE	300	TFIDF(Ext)	LR	63, 5	1, 7	29, 5	0, 9	1, 2	0, 0	2, 3	0, 1	66, 6	0, 7
	SE	Q	CVE	200	Both(Ext)	LR	50, 4	1, 5	6, 1	2, 4	2, 3	0, 8	4, 4	1, 4	65, 6	1, 1
Wicket	SO	QA	-	200	Word2Vec(Int)	LR	66.7	6.7	9.7	2.1	6.2	1.3	11.3	2.1	75.7	3.7
	SE	Q	-	200	TFIDF(Int)	NB	66.7	0.0	8.9	0.6	7.0	0.5	12.7	0.7	76.3	0.2
	SE	Q	-	200	Both(Int)	NB	66.7	0.0	8.5	1.3	6.2	0.9	11.3	1.4	75.7	0.5
Ambari	AU	Q	CWE	100	Word2Vec(Ext)	SVM	71.4	0.0	12.8	0.7	6.4	0.3	11.7	0.5	77.7	0.3
	AU	QA	-	300	TFIDF(Int)	RF	71.4	0.0	16.0	1.7	4.8	0.5	9.1	0.8	75.5	0.7
	AU	QA	CWE	100	Both(Ext)	LR	71.4	0.0	13.2	2.1	5.1	0.7	9.4	1.2	75.9	0.9
Camel	AU	Q	CAPEC	200	Word2Vec(Ext)	RF	61.1	6.5	21.2	2.7	8.6	0.4	15.2	0.6	68.9	2.3
	AU	Q	CAPEC	200	TFIDF(Ext)	RF	61.1	5.4	28.2	0.9	7.4	0.5	13.2	0.9	65.8	2.6
	AU	Q	CAPEC	300	Both(Ext)	RF	61.1	8.2	26.9	1.6	7.8	0.5	13.8	0.9	66.5	3.2
Derby	SO	Q	CVE	100	Word2Vec(Ext)	LR	57.1	2.6	21.8	1.3	18.2	1.2	27.7	1.6	65.5	1.8
	AU	Q	CWE	200	TFIDF(Ext)	NB	61.9	1.8	28.4	2.2	15.1	0.6	24.7	0.6	66.2	0.3
	SE	QA	CWE	200	Both(Ext)	NB	54.8	8.4	17.5	9.5	13.9	3.1	23.6	3.0	65.2	0.9

4.2.2 Comparing models with filtered dataset to models with unfiltered dataset

We report the summary statistics of the model with the best g-measure average for both the filtered dataset in Table 4.3 and unfiltered dataset in Table 4.4. In Table 4.3, the results show the discussion forum where the training dataset is collected from, whether we use the question (Q), or a combination of question and answers (QA), the method used to collect features from the dataset (Feature.Method), the security data source of the features (Feature.source), the size of the feature (Feature.dim), the feature used as dictionary list for the leaner algorithm (Feature.Learner), this can be either external or internal, and the learner algorithm (Learner). The metrics for Chromium across the three methods are similar - recalls between 52% and 53%, gmeasures between 66.4% and 66.6%, and precisions between 1.9% and 2.3%. These patterns are the same for Ambari, Camel, and Derby. Only Wicket has a clear best model with 83% recall and gmeasure. CAPEC and CWE are dominant security sources and AskUbuntu (AU) featured most. Both CAPEC and CWE relate to attack scenarios and weaknesses which are common in software projects unlike the CVE that describe real vulnerabilities in specific products. We cannot explain the reason for AU dominating the datasets. External features from security sources appear to produce the best and most stable models across the projects.

In comparison to unfiltered datasets (Table 4.4), the performance metrics are mostly similar across the projects. The exception is Wicket with a clear best

model recorded with a filtered dataset. This is confirmed by the hypothesis test for the Word2Vec model in Table 4.5 where the test for the metrics (recall, fscore, and gmeasure) are significant for filtered datasets with effect sizes varying between medium (0.47) and large (1.97). Another benefit of filtered datasets could be seen in the significant test under the Word2Vec model of fscore (harmonic mean between recall and precision) and g-measure (harmonic mean between recall and true negative rate). Both the fscores and gmeasures for Wicket, Ambari, Camel, Derby and Chromium (only fscore) are statistically significant with effect sizes between medium and large. The recall however is better with the unfiltered datasets. Another observation from Table 4.4 is that using externally generated features from security sources produce the best models across the projects with only Wicket as an exception. We can infer that extracting features from security sources can be beneficial for security report classification models.

Table 4.5: Hypothesis: filtered dataset vs. unfiltered dataset

		Word2Vec		TFIDF	
Project	Metric	p-value	Hedges,g	p-value	Hedges,g
Chromium	recall (pd)	0.99	−1.28	0.94	−0.60
	f-score	< 0.001	1.17	0.93	−0.41
	g-measure	0.99	−1.11	0.84	−0.19
Wicket	recall (pd)	< 0.0001	1.02	0.31	0.11
	f-score	0.021	0.47	0.10	0.18
	g-measure	< 0.0001	1.97	0.006	0.49
Ambari	recall (pd)	0.99	−0.64	0.99	−0.48
	f-score	< 0.0001	1.82	< 0.0001	0.75
	g-measure	< 0.0001	1.67	< 0.0001	0.71
Camel	recall (pd)	0.98	−0.43	0.88	−0.19s
	f-score	< 0.0001	0.60	< 0.0001	0.40
	g-measure	0.01	0.42	< 0.0001	0.89
Derby	recall (pd)	0.80	−0.18	0.99	−0.56
	f-score	< 0.0001	0.47	0.88	−0.49
	g-measure	< 0.0001	0.71	0.99	−0.91

4.2.3 Providing additional context for bug reports

We report the result of our qualitative assessments of the matches between the security sources and bug reports.

Table 4.6: Percentage of CWE, CAPEC, and CVE matches for Chromium bug reports predicted as true positive

Dataset	Category	CWE	CAPEC	CVE
Chromium	Direct	7	7	0
	Indirect	21	13	13

Table 4.6 presents the percentage of matching records from the qualitative assessment of 100 bug reports in Chromium predicted by the classification model as true positives. In summary, 7% of CWE has direct matches while 21% has indirect matches. For CAPEC, 7% has direct matches and 13% has indirect matches. For CVE, 13% has indirect matches while there was no direct match. The empty match for CVE in the sample of true positives we assessed can be explained first based on the small size of CVE records used for training the

Word2Vec model and second, because the bug reports we assessed are not attack incidents.

Table 4.7 shows examples of how bugs in Chromium predicted by the model as true positive can be tagged with related direct and indirect source(s) from our Word2Vec model. The bug report "Chrome Buffer Overflow Vulnerability" has both a CWE and CAPEC match that describes the weakness and possible attack related to it in detail. The bug report "Security cross domain thefts via CSS string property injection" has a CAPEC match that describes the related attack. Lastly, the bug report "Further restrict access of file URL." also has both a CWE and CAPEC source that can provide more security context.

Table 4.7: Examples of CWE, CAPEC, and CVE matches for Chromium bugs predicted as true positives

Dataset	Bug	CWE	CAPEC	CVE
Chromium	Chrome Buffer Overflow Vulnerability - "SaveAs" Function s...@bkav.com.vn SVRT - Bkis have just discovered vulnerability in Google Chrome 0.2.149.27 and would like to inform you with this. Here comes the report Details - Type of Issue Buffer Overflow. - Affected Software Google Chrome 0.2.149.27. - Exploitation Environment Google Chrome (Language Vietnamese) on Windows XP SP2. - Impact Remote code execution - Description The vulnerability is caused due to a boundary error when handling the "SaveAs" function. On saving a malicious page with an overly long title (¡title¿ tag in HTML) the program causes a stack-based overflow and makes it possible for attackers to execute arbitrary code.	CWE-130: If an attacker can manipulate the length parameter associated with an input such that it is inconsistent with the actual length of the input, this can be leveraged to cause the target application to behave in unexpected, and possibly, malicious ways. One of the possible motives for doing so is to pass in arbitrarily large input to the application. Another possible motivation is the modification of application state by including invalid data for subsequent properties of the application. Such weaknesses commonly lead to attacks such as buffer overflows and execution of arbitrary code...	CAPEC-242: code injection "An adversary exploits a weakness in input validation on the target to inject new code into that which is currently executing. This differs from code inclusion in that code inclusion involves the addition or replacement of a reference to a code file.	
Chromium	Security cross domain thefts via CSS string property injection. This bug is kind of annoying. The hope is that something can be done at the browser level. We really don't want to have to make sweeping changes to lots of our applications and generally burden web apps with more web browser issues. This affects Chrome Safari IE Opera and FF. But let's keep this bug Chrome-private whilst we debate what can be done (and protect our customers first) The attack involves cross-domain CSS stylesheet loading. Because the CSS parser is very lax it will skip over any amount of preceding and following junk in its quest to find a valid selector...		CAPEC-468: An attacker makes use of Cascading Style Sheets (CSS) injection to steal data cross domain from the victim's browser. The attack works by abusing the standards relating to loading of CSS: 1. Send cookies on any load of CSS (including cross-domain) 2. When parsing returned CSS ignore all data that does not make sense before a valid CSS descriptor is found by the CSS parser By having control of some text in the victim's domain, the attacker is able to inject a seemingly valid CSS string. It does not matter if this CSS string is preceded by other data. The CSS parser will still locate the CSS string. If the attacker is able to control two injection points, one before the cross domain data that the attacker is interested in receiving and the other one after, the attacker can use this attack to steal all of the data in between these two CSS injection points when referencing the injected CSS while performing rendering on the site that the attacker controls. When rendering, the CSS parser will detect the valid CSS string to parse and ignore the data that "does not make sense". That data will simply be rendered. That data is in fact the data that the attacker just stole cross domain. The stolen data may contain sensitive information, such CSRF protection tokens.	
Chromium	Further restrict access of file URL. It sucks that viewing an HTML document at a file URL compromises the confidentiality of your entire file system We should follow Firefox s lead and further restrict the privileges of file URLs	CWE-356: The software's user interface does not warn the user before undertaking an unsafe action on behalf of that user. This makes it easier for attackers to trick users into inflicting damage to their system. Software systems should warn users that a potentially dangerous action may occur if the user proceeds. For example, if the user downloads a file from an unknown source and attempts to execute the file on their machine, then the application's GUI can indicate that the file is unsafe.	CAPEC-597: Absolute Path Traversal: An adversary with access to file system resources, either directly or via application logic, will use various file absolute paths and navigation mechanisms such as ".." to extend their range of access to inappropriate areas of the file system. The goal of the adversary is to access directories and files that are intended to be restricted from their access.	

It is possible that non-security bug reports predicted by the model to be security related, are actually security related. Table 4.8 shows examples of how bugs in Chromium marked as non-security are predicted as security (false positive) and

tagged with related source(s). The bug report "IME should be disabled in password box" has a CWE match that describes the weakness related to the bug report, and the bug report "Create Master Password for Saved Passwords" has both a CWE and CAPEC match that gives the user information on the related weakness and possible attack pattern related to it.

Table 4.8: Examples of CWE, CAPEC, and CVE matches for Chromium bugs predicted as false positives

Dataset	Bug	CWE	CAPEC	CVE
Chromium	IME should be disabled in password box. Prev 1642 of 14166 Next 5 problem? 1. Open a webpage with a password box (eg. gmail.com's login page) 2. Try to enable IME and input something with IME in the password box. 3. ? IME should not be enabled and used in a password box. ? IME can be enabled and can input text in a password box. Allowing IME in a password box should be considered as a serious security issue. Think about if an IME records all input of a user and sends them to a remote server the user's password can easily be stolen in this way.	CWE-260: The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere.		
Chromium	Create Master Password for Saved Passwords. I personally do not feel safe allowing anyone access to my computer to just click maybe three buttons and see all of my passwords from a security standpoint this is A HUGE defect. What is the use of having passwords if they can all be seen so easily. So I propose creating a master password to be able to view all of the saved passwords for increased security.	CWE-922: The software stores sensitive information without properly limiting read or write access by unauthorized actors.	CAPEC-21: An adversary guesses, obtains, or "rides" a trusted identifier (e.g. session ID, resource ID, cookie, etc.) to perform authorized actions under the guise of an authenticated user or service. Attacks leveraging trusted identifiers typically result in the adversary laterally moving within the local network, since users are often allowed to authenticate to systems/applications within the network using the same identifier. This allows the adversary to obtain sensitive data, download/install malware on the system, pose as a legitimate user for social engineering purposes, and more. Attacks on trusted identifiers take advantage of the fact that some software accepts user input without verifying its authenticity...	

This shows that it is possible to use our classification model along with the Word2Vec model to provide additional context for security bug reports, as well as for discovering bug reports tagged as non-security that could actually be security related issues and providing additional context for clarity.

4.3 Threats to validity

Data sampling: We have collected a subset of the security related records in the discussion forums. These samples do not represent all types of datasets in the forums. E.g. StackOverflow contains over 80000 SR records. We have only used 2000 records. Using a bigger sample size can lead to different results. However, our validation results are proof that the sample size is a good representation.

Modelling: There are several parameters that can be tuned in the algorithms we have used. We have used the default settings for the parameters of the algorithms during model construction. In the cases where we have used specific

parameters, we have reported them in the experiment section. It is possible that tuning the parameters might lead to different results.

Generalization: The observations we have made are based on four discussion forums on StackExchange and five open source bug report datasets. We cannot claim that datasets from other forums and projects will produce the same results. Therefore, further studies will be necessary to generalize the results across other forums and software project datasets.

4.4 Conclusion

We built models by using discussion forum datasets that we have filtered with security sources, and compare their performances to unfiltered datasets from the forum. By validating the models on five popular security bug report datasets, we establish that features extracted from security sources can be useful to build security report models. We also find that filtered datasets do not produce models with higher recalls to unfiltered ones, however, they produce models that are better in fscore and gmeasure. Lastly, our models' performances are competitive to state-of-the-art studies.

We investigated using public security sources such as CWE, CAPEC, and CVE to provide additional context for Chromium bug reports predicted by the model to be true positive or false positive. We assess the CWE, CAPEC, and CVE recommendations from our Word2Vec models for a sample of Chromium bug reports predicted as true positive. We find 7% direct matches for CWE, 7% direct matches for CAPEC, 21% indirect matches for CWE, 13% indirect matches for CAPEC, and 13% indirect matches for CVE. In addition, we assessed a sample of bug reports predicted by the classification model to be false positive for hidden security implications that can be contextualized using public security sources. We found examples where CWE and CAPEC could give valuable information on the security threats related to the bug report, as well as suggestions on how to mitigate them.

In conclusion, our results demonstrate a useful layer of validation for security classification models, by providing additional security context information.

Chapter 5

Case study 2: Contextualizing Public Forum Discussions Using Public Security Data Sources

In this case study, we aim to answer RQ2: How can public information sources be used to improve security in online discussion forums?

Online discussion forums have become de facto knowledge factory for solving software related problems. Unfortunately, solutions from public discussion forums are not security hardened, and have been shown to have security implications [1, 4, 17]. Acar et al. [1] showed that the use of StackOverflow by developers leads to insecurity as developers copy-paste vulnerable solutions. We argue that providing additional context to questions on discussion forums and bug reports by using security information sources from publicly disclosed cybersecurity vulnerabilities (CVE), common weaknesses (CWE), and attack patterns (CAPEC) can improve this situation and make stakeholders better evaluate security related reports. Thus, users of public information resources such as StackOverflow can better evaluate the security implications of their discussions or the solutions they are adapting in their environment if this information can be mapped to existing security information sources.

Our hypothesis is that security information sources can provide an additional validation layer of security classification for public information sources. In this case study, we use a word embedding Word2Vec model to provide mappings between discussion forum questions and security sources in order to answer RQ2.

5.1 Experiment setup and modelling approach

To answer RQ2, we use a Word2Vec model that we have trained on each of the security sources. We then select the first 100 security related (SR) records from StackOverflow and ServerFault. For each record, we use the mean vector embedding from the Word2Vec model described in 3.3.2 and then perform a cosine similarity to the mean vector embedding of each record from the security source. Using a cosine of 0.5 as our threshold, we collect the top five most similar security sources to a particular forum question. We also evaluate collected non-security related (NSR) records from StackOverflow and ServerFault that have a cosine similarity of 0.5 for any security source. We collect the top five most similar security sources for each particular NSR question.

We then provide a qualitative assessments of these mappings to external security sources to identify whether they are meaningful or not. When assessing whether a source reasonably matches a discussion, we use the method described in 2.6.2. Two raters have independently evaluated the 100 SR records and compared results in addition to resolving disagreements. A record has one to five suggested matches. We record a one for a category when at least one of the five matches is marked with this category. For the NSR records, only one rater evaluated them.

5.2 Results & Discussion

We report the result of our qualitative assessments of the matches between the security sources and discussion forum questions.

Table 5.1: Percentage of CWE, CAPEC, and CVE matches in StackOverflow and ServerFault discussion questions

Forum	Category	CWE	CAPEC	CVE
StackOverflow	Direct	12	10	0
	Indirect	44	54	14
ServerFault	Direct	14	8	0
	Indirect	20	14	18

Table 5.1 presents the percentage of matching records from qualitative assessment of 100 security-related questions in both StackOverflow and ServerFault. In summary, for StackOverflow, 12% of CWE has direct matches while 44% has indirect matches. For CAPEC, 10% has direct matches and 54% has indirect matches. For CVE, 14% has indirect matches while there was no direct match. For ServerFault, 14% of CWE has direct matches while 20% has indirect matches. For CAPEC, 8% has direct matches and 14% has indirect matches. For CVE, 18% has indirect matches while there was no direct match.

The empty match for CVE in the sample of SR we assessed can be explained first based on the small size of CVE records used for training the Word2Vec model and second, because the discussion forum questions we assessed are not attack incidents. Tables 5.2 and 5.3 present specific examples of questions with direct and indirect matches to the security sources. The matches provide, in some

cases, examples of incidents (CVE) that can be relevant to the question. For example, the user with question-Id 934731 is looking to log all account management activities for at least 6 months. However, CVE-2020-15095 in Table 5.3 presents a real world information exposure vulnerability in npm cli through log files that can help the user think through certain mitigations while implementing this feature. CAPEC-93: Log Injection-Tampering-Forging, further amplifies the pattern of attack. Thus, the stakeholder can think of mitigation in advance such as digitally signing every record in the log file in order to detect tampering.

In some other cases, the matches only provide information to users. For example, the questions id - 1057084 (fixing weak TLS), 104870 (Memory Pressure Protection feature) in Table 5.3 and 63605452 (XML injection), 28606689 (Screen Capture in Android), and 332365 (SQL injection) all have CWE and CAPEC matches that describe and discuss the implications of the security questions.

Table 5.2: Examples of CWE, CAPEC, and CVE matches to security related StackOverflow discussion questions

Q-ID	Question	CWE	CAPEC	CVE
63605452	How to prevent XML injection - I got a vulnerability report. XML is injected in the URL "XInclude". I'm trying to find a validation to prevent the XML to be executed. My web application is built using Visual Studio C# with webforms. I was thinking to validate this from the web.config or IIS. I'm not sure if I have to add code to validate or parse the XML...	CWE-611: Improper Restriction of XML External Entity Reference - The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output...	CAPEC-250: An attacker utilizes crafted XML user-controllable input to probe, attack, and inject data into the XML database, using techniques similar to SQL injection. The user-controllable input can allow for unauthorized viewing of data, bypassing authentication or the front-end application for direct XML database access, and possibly altering database information	
28606689	How to prevent Screen Capture in Android - Is it possible to prevent the screen recording in Android Application? I would like to develop an Android Secure Application. In that I need to detect screen recording software which are running background and kill them. I have used SECURE FLAG for prevent screenshots. But I dont know is it possible to prevent Video capturing of Android Screen also. Let me know how to prevent screen capturing (video / screenshots)...		CAPEC-648: Collect Data from Screen Capture - An adversary gathers sensitive information by exploiting the system's screen capture functionality. Through screenshots, the adversary aims to see what happens on the screen over the course of an operation. The adversary can leverage information gathered in order to carry out further attacks...	
332365	How does the SQL injection from the "Bobby Tables" XKCD comic work? What does this SQL do: Robert'); DROP TABLE STUDENTS; – I know both ' and – are for comments, but doesn't the word DROP get commented as well since it is part of the same line?...	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component...		

Table 5.3: Examples of CWE, CAPEC, and CVE matches to security related ServerFault discussion questions

Q-ID	Question	CWE	CAPEC	CVE
934731	Using auditd and retaining log files for 6 months. - Find a way to log all account management activities (e.g., account creation, modification, deletion, etc.) on an Ubuntu 16.04 LTS server and retain the logging information for at least 6 months...		CAPEC-93: Log Injection-Tampering-Forging - This attack targets the log files of the target host. The attacker injects, manipulates or forges malicious log entries in the log file, allowing them to mislead a log audit, cover traces of attack, or perform other malicious actions. The target host is not properly controlling log access. As a result tainted data is resulting in the log files leading to a failure in accountability, non-repudiation and incident forensics capability.	CVE-2020-15095: versions of the npm cli prior to are vulnerable to an information exposure vulnerability through log files the cli supports urls like protocol hostname path the password value is not redacted and is printed to stdout and also to any generated log files
1057084	How to fix Weak TLS 1.2 Encryption - I have a requirement to disable below weak TLS ciphers in Windows Server 2016. I tried to reasearch and it says "The Microsoft SCHANNEL team does not support directly manipulating the Group Policy and Default Cipher suite locations in the registry" Please advise. Thank you in advance...	CWE-326: Inadequate Encryption Strength - A weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of succeeding using current attack methods and resources.		
104870	Memory Pressure Protection Feature for TCP Stack - Provided by Microsoft Security Update KB967723 - We've been having a lot of funky issues with some of our web based applications that allow clients to submit lot of image files to our servers. Lots of ports are used in the process... There doesn't appear to be a pattern and sometimes it works and other times is doesn't. Typically we've noticed it when server is under load.I'm curious what others think about this MPP and any issues that you may have experienced from it...	CWE-406: Insufficient Control of Network Message Volume (Network Amplification) - In the absence of a policy to restrict asymmetric resource consumption, the application or system cannot distinguish between legitimate transmissions and traffic intended to serve as an amplifying attack on target systems. Systems can often be configured to restrict the amount of traffic sent out on behalf of a client, based on the client's origin or access level...		

Table 5.4 and 5.5 present examples of non-security related questions posted on StackOverflow and ServerFault, where our Word2Vec model has suggested CWE and CAPEC matches that can provide the user with additional context regarding the possible security vulnerabilities. Question ID 66540743 (REST - GET - Passing secret value) in Table 5.4 gives an example of a possible weakness that can take place if the user does not apply the right measures for avoiding it. Question ID 1056340 (Pfsense External IP to Internal Server) in Table 5.5 has both a CWE match and CAPEC match that discusses the possible security implications related to this question.

Table 5.4: Example of CWE, CAPEC, and CVE matches to a non-security related StackOverflow discussion question

Q-ID	Question	CWE	CAPEC	CVE
66540743	REST - GET - Passing secret value I am designing a REST API for getting a gift card balance I choose GET HTTP verb to be used for it But then I realized that I need to pass the PIN number of the gift card for which balance needs to be fetched I am a bit puzzled with the design I can think of two approaches Use HTTP GET and pass PIN as a custom HTTP header X-GC-PIN URI giftcards gift-card-number PIN is not a meta-data so I am reluctant to pass it as a HTTP header Use HTTP POST and pass PIN in the body And URI giftcards gift-card-number balance But since this is a fetch call I am reluctant to use POST How I should approach this design problem Is any of the above is more suitable or is there another way	CWE-614: The Secure attribute for sensitive cookies in HTTPS sessions is not set which could cause the user agent to send those cookies in plaintext over an HTTP session		

Table 5.5: Example of CWE, CAPEC, and CVE matches to a non-security related ServerFault discussion question

Q-ID	Question	CWE	CAPEC	CVE
1056340	Pfsense External IP to Internal Server I am currently searching for some help regarding firewall network traffic Mind you I am not a guru by any means so some terms may not be proper and I will do my best to explain Diagram 1 Currently I host a game server This game server is setup as described and can be viewed from the Network Diagram Image below Server ip is 10 1 0 120 my public is is 67 x x x I have port forwarded all ports 2502 2512 and 27018 This allows users to connect into my network and play the game This is a security risk as my external ip is visible and there is nothing securing my line other than my ability to lock it down There is nothing wrong with this connection and it performs great Well until This is where things get a bit sketchy My current public ip of 67 x x x is visible to everybody and their mother on the internet so if someone ddos attacks me I no longer have access to the internet and neither do the 50-80 people connecting into my server...	CWE-941: Attackers at the destination may be able to spoof trusted servers to steal data or cause a denial of service. There are at least two distinct weaknesses that can cause the software to communicate with an unintended destination: If the software allows an attacker to control which destination is specified, then the attacker can cause it to connect to an untrusted or malicious destination... As another example, server-side request forgery (SSRF) and XML External Entity (XXE) can be used to trick a server into making outgoing requests to hosts that cannot be directly accessed by the attacker due to firewall restrictions..	CAPEC-590: An adversary performing this type of attack drops packets destined for a target IP address. The aim is to prevent access to the service hosted at the target IP address.	

5.3 Threats to validity

Data sampling: We have collected a subset of the security related records in the discussion forums. These samples do not represent all types of datasets in the forums. E.g. StackOverflow contains over 80000 SR records. We have only used 100 records. Using a bigger sample size can lead to different results.

Generalization: The observations we have made are based on two discussion

forums on StackExchange. We cannot claim that datasets from other forums and projects will produce the same results. Therefore, further studies will be necessary to generalize the results across other forums and software project datasets.

5.4 Conclusion

We have investigated using public security sources such as CWE, CAPEC, and CVE to provide additional context for security discussions on StackExchange forums. We argue that such context can further clarify questions and provide examples of weaknesses, attack patterns, and real-world scenarios related to the question.

We assess the CWE, CAPEC, and CVE recommendations from our Word2Vec models for a sample of security related StackOverflow and ServerFault records. We find 12% - 14% direct matches for CWE, 8% - 10% direct matches for CAPEC, 20% - 44% indirect matches for CWE, 14% - 54% indirect matches for CAPEC, and 14% - 18% indirect matches for CVE. In addition, we have assessed non-security related questions from StackOverflow and ServerFault for hidden security implications that can be contextualized using public security sources. We found examples where CWE and CAPEC could give the user valuable information on the security threats related to their questions, as well as suggestions on how to mitigate them.

In conclusion, our results demonstrate a useful model for providing additional security context for public discussion forum posts related to security issues, as well as promising results for finding hidden security implications in non-security related discussion forum posts.

Chapter 6

Improving security in the developer's development environment

In this thesis, we have not answered RQ3. However, we describe the idea behind the RQ which can be pursued in future work. RQ3 is as follows: How can public information sources be used to improve security in the developer's development environment?

To properly validate RQ3, we would collect datasets from code sharing platforms or public code bases. Figure 6.1 gives an overview of the suggested approach for validating RQ3. The API already has interfaces for two code sharing platforms (StackExchange and ProgramCreek), but an interface for code bases (e.g. Github [27]) would be a good extension. Our approach is based on the idea presented by Scandariato et al. [52]. The idea is to collect a dataset consisting of vulnerable code and clean code. To do his, we can run a static analysis tool on several code bases or code sharing platforms, as well as collect vulnerable examples by mining the examples from e.g. CWE, using the interfaces in the API described in section 3.1.2, or CERT. We can then train a vulnerability classification model on this code dataset using the modelling approach from chapter 4 shown in Figure 4.1.

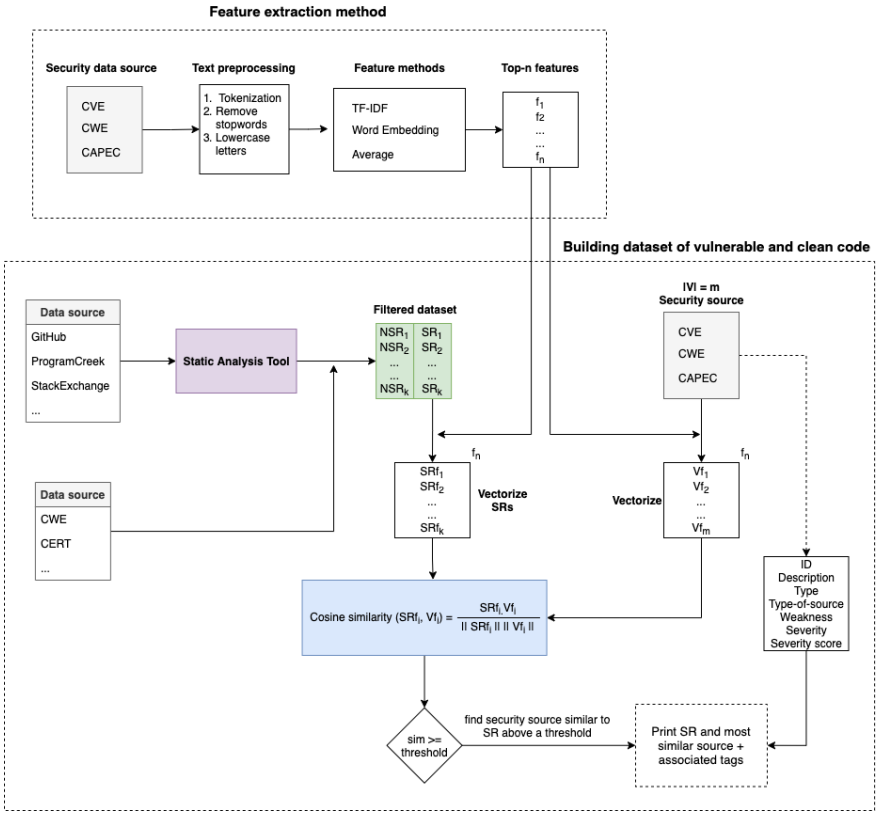

Figure 6.1: Approach for collecting datasets consisting of vulnerable and clean code

The final classification model from this approach will then be integrated into a developers IDE (e.g. Eclipse or IntelliJ). The idea is that when a developer for example copies code from a discussion forum, when pasting it into the IDE, the model will first process the code sample and predict whether is is security vulnerable or not. In addition, the developer should be given suggestions of public security source matches to give the code additional security context. To do this, we will use a Word2Vec model trained on CVE, CWE and CAPEC code examples to identify closely matched context.

Chapter 7

Discussion

In this chapter, we discuss some of the design choices and challenges we faced when developing the API, as well as the results of the case studies conducted in this thesis.

7.1 API Development

When developing the API, we had to decide on what types of sources to build interfaces for as a start. Because of the time limit, we decided to include interfaces for three types of sources; public discussion forums, public security sources and code sharing platforms. This gives a solid foundation for the API, and allows for further extensions later on, for instance adding interfaces for other discussion forums, public information sites, expert blogs, code sharing platforms or public code bases. Currently, the API has interfaces for two code sharing platforms, namely StackExchange and ProgramCreek. The idea behind this was to create the groundwork for answering RQ3 in the future as described in chapter 6. The interfaces for the public security sources and code sharing platforms uses web crawlers built specifically for each website. The issue with this is that any sort of change or update to the HTML tags on the site must also be addressed in our API implementation. However, as these websites did not have any sort of API for collecting records, this was the best way to do it. This also led to some challenges, as not all pages on a website had the same layout, so we had to implement workarounds for these problems as we encountered them. Code example 7.1 shows an example of this, where we implemented the crawler to skip rejected CVE records.

```
1 // skip CVE if it is rejected
2 if(desc.asText().contains("** REJECT **")){
3     number++;
4     continue;
5 }
```

Listing 7.1: Example of a workaround in CVE web crawler

We decided early on to use the two feature extraction methods TF-IDF and Word2Vec, because these methods have been shown to have good results on

similar tasks in several papers [46, 44, 67]. However, the API is built in such a way that it should it be easy to add new feature extraction methods. This is further described in chapter 10. All feature sets, word vectors, datasets and classification models are stored in files, as the classification algorithms in Weka and Deeplearning4j all use files as input. We encountered some challenges when writing Word2Vec vectors to a file for the validation dataset Chromium, as some of the bug reports only containing one word in the description were not properly written in a vectorized form. Instead the number vectors were replaced by the replacement character (Unicode U+FFFD). This resulted in an exception error when trying to run the Convolutional Neural Network classification model on the file. To fix this, we had to clean all vectorized Chromium files, and replace the odd symbols with 0.0 instead. A method for fixing this problem was created in the DataPreProcess class.

7.2 Case Study Results

7.2.1 Contextualizing Bug Reports Using Public Security Data Sources

The case study described in chapter 4, investigated RQ1 by creating classification models from public discussion forum datasets and public security datasets, as well as by using a Word2Vec model to find similar public security sources for Chromium bugs predicted by the best performing classification model to be true positive or false positive.

The results of the classification models were compared to results of state-of-the-art studies, and showed clear competitive results. When comparing the filtered dataset results to the unfiltered dataset results, we could see that filtered datasets generally performed slightly better than unfiltered datasets with a few exceptions. The explanation for this could be that even though the unfiltered datasets are not extracted using any of the feature extraction methods described in Figure 3.3, the SRs could still contain terms that give them a high cosine similarity, resulting in a model being able to predict them correctly. We also saw that using externally generated features from security sources produced the best models across the projects with only one exception for the Wicket dataset. We can therefore infer that extracting features from security sources can be beneficial for security report classification models. There are several factors in this case study that could lead to different results. Firstly, the collected datasets do not represent all types of datasets that can be collected from forums on StackExchange. StackExchange contain over 80000 security-related records, but we have only used 2000 records. Using a bigger sample or a different threshold for filtering datasets can lead to different results. However, our validation results are proof that the sample size is a good representation. Secondly, datasets from other forums and software projects might produce different results, so further studies will necessary to generalize the results across forums and project datasets. Lastly, we have used the default settings for the parameters of the classification algorithms during model construction. Tuning the parameters could possibly lead to different results.

The results of the qualitative assessment of 100 bug reports on Chromium pre-

dicted by the classification model as true positives, demonstrated that providing additional security context for bug reports can be a useful layer of validation for security classification models. We assessed the CWE, CAPEC, and CVE recommendations from our Word2Vec models for a sample of Chromium bug reports predicted as true positive. We found 7% direct matches for CWE, 7% direct matches for CAPEC, 21% indirect matches for CWE, 13% indirect matches for CAPEC, and 13% indirect matches for CVE. In addition, we found examples of false positives where the model suggested public security sources that explained the possible security implications of the reported bug. The empty match for CVE in the sample of bug reports we assessed can be explained first based on the small size of CVE records used for training the Word2Vec model and second, because the bug reports we assessed are not attack incidents. There are factors that could possibly lead to different results for this qualitative assessment. For instance, the sample of Chromium bugs in the dataset do not represent all types of bugs in the Chromium project. In addition, many of the security related bug reports in the Chromium dataset only have a one-word description of "Security", which is correctly predicted by the classification model, but is not possible to find any public security source matches for as it does not include any context for the bug.

7.2.2 Contextualizing Public Forum Discussions Using Public Security Data Sources

The case study described in chapter 5, investigated RQ2 by using a word embedding Word2Vec model to provide mappings between public security sources and StackOverflow and ServerFault forum questions. The result of our qualitative assessments of the matches between the security sources and discussion forum questions were promising. For StackOverflow, 12% of CWE has direct matches while 44% has indirect matches. For CAPEC, 10% has direct matches and 54% has indirect matches. For CVE, 14% has indirect matches while there was no direct match. For ServerFault, 14% of CWE has direct matches while 20% has indirect matches. For CAPEC, 8% has direct matches and 14% has indirect matches. For CVE, 18% has indirect matches while there was no direct match. The empty match for CVE in the sample of SRs we assessed can be explained first based on the small size of CVE records used for training the Word2Vec model and second, because the discussion forum questions we assessed are not attack incidents. There are factors that could possibly lead to different results for this qualitative assessment. For instance, the sample of discussion forum questions do not represent all types of questions on the various forums. Collecting different datasets might lead to different results.

Chapter 8

Related Work

Related works were surveyed on public information security sources, bug report classification, linking knowledge units, as well as some relevant work on machine learning and natural text processing that could benefit the bug report classification.

The systematic literature review was carried out based on the snowballing methodology described by Wohlin [62]. The approach consists of a start set of papers and several iterations for finding papers that will be included in the final version. The iterations include two steps: backwards snowballing which looks at the references of the considered paper, and forwards snowballing for identifying new papers citing the paper being considered. For the literature review, three papers regarding public information security sources and two papers regarding linking knowledge units were selected as the start set. Several iterations were conducted until no more relevant papers were found. The final version was a set of 37 papers covering four relevant topics.

8.1 Public Information Security Sources

Sauerwein et al. [51] conducted a triangulation study to identify and analyze public information security data sources, as well as introducing a taxonomy to classify and compare these data sources. Their investigations showed that research and practice rely on a large variety of heterogeneous information security data sources, which makes it difficult to integrate and use for information security and risk management processes.

Felderer et al. [16] propose a framework for security data extraction, processing and application. The framework consists of a security data collection component and an analysis component, as well as a security knowledge generation component.

Wijayasekara et al. [61] discusses existing bug data mining classifiers, and present an analysis of vulnerability databases showing the necessity to mine common publicly available bug databases for hidden impact vulnerabilities.

Pletea et al. [47] evaluated the presence and atmosphere surrounding security-related discussions on GitHub from discussions around commits and pull requests.

Ohira et al. [43] introduced a dataset of high impact bugs that was created by manually reviewing four thousand issue reports in four open source projects.

Islam et al. [29] studies 2716 high-quality posts from Stack Overflow and 500 bug fix commits from Github about five popular deep learning libraries to understand the types of bugs, root causes of bugs, impacts of bugs, bug-prone stage of deep learning pipeline as well as whether there are some common antipatterns found in this buggy software.

Wei et al. [59] proposes a semantics-aware approach for warning prioritization in Link, a static analyzer for detecting bugs/issues in Android apps, called OASIS. OASIS combines program analysis and NLP techniques to recover intrinsic links between the Link warnings and the user complaints of an app.

8.2 Bug Report Classification

Wu et al. [64] propose an automated data labeling approach based on iterative voting classification for identifying security bug reports.

Tyo [57] explored the distribution and characteristics of security vulnerabilities in bug tracking systems, as well as data analytics approaches for automatic classification of security bug reports. They used supervised learning algorithms as well as a novel unsupervised machine learning approach. Their results showed most consistently good results from the learning algorithm Naive Bayes across all datasets used.

Peters et al. [46] proposes a framework, FARSEC, for filtering and ranking bug report for reducing the presence of security bug reports that removes non-security bug reports containing security related keywords before building a prediction model. They demonstrate that FARSEC improves the performance of text-based prediction models for security bug reports in 90% of cases.

Gegick et al. [23] developed an approach to identify mislabeled security bug reports that applies text mining on natural-language descriptions of bug reports to train a statistical model.

Shu et al. [56] aims to aid software developers to better classify bug reports that identify security vulnerabilities as security bug reports through parameter tuning of learners and data pre-processor.

Goseva-Popstojanova et al. [25] focuses on automated classification of software bug reports to security and not-security related using both supervised and unsupervised approaches. Their results showed that the supervised learning slightly outperforms the unsupervised learning, at the expense of labeling the training set. In general, the results showed that datasets with more security information lead to better performance.

Wijayasekara et al. [60] presents a hidden impact bug identification methodology by means of text mining bug databases. Wright et al. [63] conducted an

experiment using the MySQL bug report database to estimate the number of misclassified bugs yet to be identified as vulnerabilities.

Gantzer [22] focuses on exploring techniques that have potential to improve the performance of automated classification of software bug reports as security or non-security related. They use feature selection, clustering and deep learning for classification.

Shu et al. [55] proposes a method, SWIFT, that combines learner hyperparameter optimization and pre-processor hyperparameter optimization for distinguishing security-related bug reports in a product's bug database. Their results show that their approach achieves better performance in a fast way than existing state-of-the-art method.

Jiang et al. [30] proposes a novel approach, LTRWES, that incorporates learning to rank a word embedding into the identification of SBRs. Their results show that the proposed method outperforms state-of-the-art method.

Zhou et al. [69] describe an efficient automatic vulnerability identification system for tracking large-scale projects in real time using natural language processing and machine learning techniques.

Kudjo et al. [34] introduces a new approach for vulnerability classification using term frequency and inverse gravity moment (TF-IGM). Their results show that TF-IGM outperforms the benchmark method across the applications used in the study.

Chawla et al. [9] presents an automated technique for bug labeling using TF-IDF and LSI. Their results showed better results from using LSI along with TF-IDF compared to using TF-IDF alone.

Das et al. [13] proposes a learning based approach to identify security and performance bug reports addressing class-bias and feature-skew phenomenon.

Mostafa et al. [41] proposes an automatic approach to identify security bug reports in open bug repositories using semi-supervised learning and keyword-based pre-filtering based on keywords mined from existing security related textual description in the CVE. Their results show that the approach outperforms the best baseline approaches.

Scandariato et al. [52] presents an approach based on machine learning to predict which components of a software application contain security vulnerabilities by text mining the source code of the components.

Oyetoyan et al. [44] investigates whether a generic text classification model can be developed for classifying security related messages in software development project communications. Their approach showed improvement in 75% cases over a state-of-the-art prediction model for security bug reports.

8.3 Linking Knowledge Units

Xu et al. [66] formulates the problem of predicting semantically linkable knowledge units as a multiclass classification problem, and solve the problem using deep learning techniques.

Xu et al. [65] proposes a query-focused multi-answer-posts summarization task for a given technical question, AnswerBot. Their user study results showed that answers generated by their approach were relevant, useful and diverse.

Bogdanova et al. [6] aims to detect semantically equivalent questions in online user forums by performing an extensive number of experiment using data from two different Stack Exchange forums, and compare standard machine learning methods with a convolutional neural network. Their results showed that the convolutional neural network with in-domain word embeddings achieved high performance even with limtied training data.

Zhang et al. [68] proposes an automated approach to duplicate question detection that takes a new question as input and detects potential duplicates of this question considering multiple factors. Their results show that the approach improves the recall-rate of 40.63% compared to the standard search engine of Stack Overflow.

Yang et al. [67] proposes a novel approach for similar bug recommendation using traditional information retrieval techniques and a word embedding technique. The results show that their approach improves the performance of similar bug recommendation system, NextBug, significantly and substantially.

Li et al. [36] conducted an empirical study to explore the characteristics of a large number of links within popular Python projects in GitHub.

Cai et al. [8] proposes a tool, AnswerBot, which enables to automatically generate an answer summary for a technical problem based on Stack Overflow. The results showed thath the answer summaries generated by AnswerBot were more relevant, useful and diverse than Google and Stack Overflow search engine.

8.4 Machine Learning and NLP

Lopez et al. [37] explains the basics of Convolutional Neural Networks (CNNs), the different variations and how they have been applied to Natural Language Processing (NLP).

F. Sebastiani [54] discusses the main approaches to text categorization that fall within the machine learning paradigm.

Allahyari et al. [2] describe several of the most fundamental text mining tasks and techniques including text pre-processing, classification and clustering.

Kenter et al. [32] investigates whether determining short text similarity is possible using only semantic features by using word embeddings.

Y. Kim [33] shows that a simple convolution neural network with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks.

Chapter 9

Conclusion

Software vulnerabilities are increasing security focus as critical systems become more dependent on complex software systems. Solutions may contain security risks that could have been avoided if the design choices were analyzed by using public information security data sources. Because of this, easily processable and up-to date security information should be available for system architects and developers during system design, development, and bug resolution. It has been observed that public discussion forums such as StackOverflow contain more recent and relevant comments on current technologies than any textbook or research article [20], and these public discussion forums are heavily used for solving software related problems. However, solutions copied from these forums can often be found to have security implications when they are copied directly into the production environment [1, 4, 17]. In this thesis, we have developed an API that provides developers with the ability to interface public security information sources and public discussion forums, extract data from these sources, populate them into existing structured expression languages, build machine learning classification models from these sources, and provide mapping between public security sources and public discussion forum posts.

RQ1 questioned how public security information sources can be used to improve bug report security classification models. The case study described in chapter 4 approached RQ1 by building classification models from public discussion forum datasets and public security datasets, as well as by using a Word2Vec model to find similar public security sources for Chromium bug reports predicted by the best performing classification model to be true positive or false positive. The results showed that features extracted from security sources can be useful for building security report models, as the models' performances showed results that are competitive to state-of-the-art studies. In addition, the matches for Chromium bug reports found by the Word2Vec model showed that the model could give valuable information on the security threats related to the bug report, as well as suggestions on how to mitigate them. In conclusion, the case study demonstrated a useful layer of validation for security classification models, by providing additional security context information.

RQ2 questioned how public security information sources can be used to improve

security in online discussion forums. The case study described in chapter 5 investigated RQ2 by using a word embedding Word2Vec model to provide mappings between public security sources and StackOverflow and ServerFault forum questions. The results of the qualitative assessment conducted by the case study, showed a promising amount of matches for the security related questions. In addition, the model found examples where CWE and CAPEC could give the user valuable information on the security threats related to their non-security related questions, as well as suggestion on how to mitigate these threats.

RQ3 questioned how public security information sources can be used to improve security in the developer's development environment. In this thesis, we have not answered RQ3. However, the idea behind the RQ which can be pursued in future work is described in detail in chapter 6. The idea is to collect datasets consisting of vulnerable and clean code examples using a static analysis tool, and train a classification model using the collected datasets following the modelling approach in chapter 4. The final classification model will be integrated into an IDE, and used to predict whether pasted code into the IDE is security vulnerable or not. In addition, it will give the developer suggestions of related public security source matches to give the developer additional security context for the vulnerable code.

Chapter 10

Future Work

The API could be extended in several different ways. It would be beneficial to add interfaces to other discussion forums, public information sites, expert blogs or code sharing platforms as this would give the user more opportunities to collect datasets and use the model to find similar public security sources. In addition, it would be interesting to add more feature extraction methods. A feature extraction method that would be interesting to explore in this context is Doc2Vec proposed in [35]. Doc2Vec is an extension of the Word2Vec word embedding model that is based on whole sentences (or documents) instead of singular words in a context. It would be very interesting to see how classification models and similarity calculations using this model would perform compared to TF-IDF (see 3.3.1) and especially Word2Vec (see 3.3.2). The collection of feature words for the classification models could also use some refinement. Currently, when collecting feature words using Word2Vec, we only collect the top n feature words from the model with no particular logic behind it. Although we have seen good results from doing it this way, it could possibly be beneficial to come up with a method to extract these words in an intelligent way.

Figure 10.1 shows a possible way to combine the models we have developed in this thesis for the purpose of refining the classification model. The idea is to use the Word2Vec model as a validation model, where the classification result (SR or NSR) is passed through the Word2Vec model. Using a certain threshold, we can check whether additional context from CWE, CAPEC, or CVE can be derived. If a classification result is NSR but has a context from the Word2Vec model, we can conclude that it is a SR (FN $\rightarrow$ TP). Similarly, if a classification result is SR and does not have a context, then this may probably be a NSR (FP $\rightarrow$ TN). However, such has to be re-inspected. In addition, it can also be that results from the Word2Vec model can be used during training for re-training models.

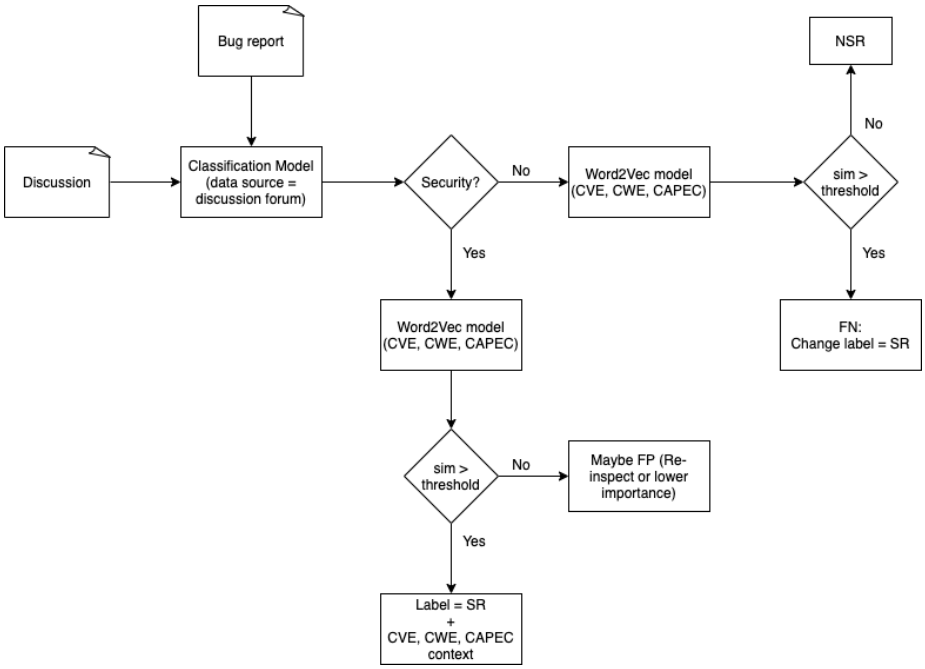

Figure 10.1: Using a combination of the classification model and word embedding model for validation

To properly validate RQ3, we would collect datasets from code sharing platforms or public code bases. The API already has interfaces for two code sharing platforms (StackExchange and ProgramCreek), but an interface for code bases (e.g. Github [27]) would be a good extension. Our approach is based on the idea presented by Scandariato et al. [52]. The idea is to collect a dataset consisting of vulnerable code and clean code. To do his, we can run a static analysis tool on several code bases or code sharing platforms, as well as collect vulnerable examples by mining the examples from e.g. CWE or CERT. We can then train a vulnerability classification model on this code dataset. The final classification model from this approach will then be integrated into a developers IDE (e.g. Eclipse or IntelliJ). The idea is that when a developer for example copies code from a discussion forum, when pasting it into the IDE, the model will first process the code sample and predict whether is is security vulnerable or not. In addition, the developer should be given suggestions of public security source matches to give the code additional security context. To do this, we will use a Word2Vec model trained on CVE, CWE and CAPEC code examples to identify closely matched context.

Appendix A

Source code

The source code for the API is available at this URL: `https://github.com/anjfs/Security-Information-API`.

Appendix B

Research Paper: Utilizing public repository to contextualize security report classification models

This research paper was written in collaboration with my supervisor, Tosin Daniel Oyetoyan, and comprises parts of the thesis. It has been submitted to the 7th International Conference on Data Mining (DTMN 2021).

B.1 Abstract

It has been remarked that modern software engineering is evolving so fast that public forums contain more relevant and recent comments on current technologies than any textbook or research article. Public forums are now de-facto knowledge factory for solving software related problems. Unfortunately, not every solution is security hardened. Solutions copied from public discussion forums such as stackoverflow to production environment have been shown to have security implications.

We investigate whether we can provide additional context to both discussion forum contents and bug reports by using public security sources such as CVE, CWE, and CAPEC. We construct classification models with discussion forum dataset filtered by and built with features from public security sources. We then validate our model using commonly used bug report dataset. Finally, using similarity measures and word embedding technique, we map discussion questions that are related to security sources.

Our results show that features extracted from public security records produce stable models. By performing a qualitative assessment of a sample of stackoverflow and serverfault records that are mapped to public security sources from word embedding models, we find 12% - 14% direct matches for CWE, 8% - 10%

direct matches for CAPEC, 20% - 44% indirect matches for CWE, 14% - 54% indirect matches for CAPEC, and 14% - 18% indirect matches for CVE. Our result demonstrates additional layer for validating security classification models.

B.2 Introduction

It has been remarked that modern software engineering is evolving so fast that public forums contain more relevant and recent comments on current technologies than any textbook or research article [20]. Public forums are now de-facto knowledge factory for solving software related problems. Unfortunately, not every solution is security hardened. Solutions copied from public discussion forums such as StackOverflow to production environment have been shown to have security implications [1, 4, 17]. Acar et al. [1] showed that the use of StackOverflow by developers leads to insecurity as developers copy-paste vulnerable solutions. We argue that providing additional context to questions on discussion forums and bug reports by using security information sources from publicly disclosed cybersecurity vulnerabilities (CVE), common weaknesses (CWE), and attack patterns (CAPEC) can improve this situation and make stakeholders better evaluate security related reports. In addition, it can support vulnerability models built using source code [70, 52]. Thus, users of public information resources such as StackOverflow can better evaluate the security implications of their discussions or the solutions they are adapting in their environment if this information can be mapped to existing security information sources.

Our hypothesis is that security information sources can provide additional validation layer of security classification for bug reports and public information sources. To address our hypothesis, we first build a classification model from both public discussion forum dataset and public security dataset. We then validate our models using open source bug report dataset. Our justification for choosing the bug reports as ground truth is that software development artifacts and discussion forums have been shown to be associated [29, 58, 48]. Second, we use word embedding model to provide mappings between discussion forum questions and security sources. We provide answers to two questions:

1. What is the performance of vulnerability classification model built with public information sources?
2. Can we gain additional context for discussion forums from public security sources?

The rest of the paper is structured as follows: Section B.5 discusses related studies, Section B.3 provides the details of our methodology. Section B.4 presents our results and provide discussions. Section B.6 discusses the threats to the validity of our results. We conclude the paper in Section B.7.

B.3 Methodology

In this section, we describe our dataset collection approach, experiment methods, and research question analysis.

B.3.1 Dataset collection

In this study, we have used data from public discussion forums, public security sources, and open source bug dataset.

Public security sources

We collected security information from three public security sources; the publicly disclosed vulnerabilities (CVE) [1], common weaknesses enumeration (CWE)[2], and common attack pattern enumeration and classification (CAPEC)[3]. As shown in Figure B.1, we collect the features for filtering dataset from discussion forums (see B.3.1) and also training machine learning models from these sources. We have collected 5000 records from 2019 and 2020 from CVE due to computational limitations and from CWE and CAPEC, we collected all available records.

Public discussion forums:

We collected posts from four discussion forums namely StackOverflow (SO), AskUbuntu (AU), SoftwareEngineering (SE), and ServerFault (SF) where $tag = security$ for security related posts (SRs) and $tag != security$ for non-security related posts (NSRs) as illustrated in Figure B.1. We have used the stackexchange API v2.2[4] to collect the data. We separate the SR dataset into two categories - filtered and unfiltered dataset. Filtered dataset is collected by performing a similarity score between a given post and each of the reports in a given security source document (CVE, CWE or CAPEC). We include a discussion if the cosine similarity score to a particular security source is above a certain threshold. For this work, we have used a threshold of 0.4 in order to obtain enough SR records from forums with low security related discussions. The cosine similarity scores are calculated using three different information retrieval methods; TF-IDF, word embedding, and taking the average of the scores from the two methods. In total, we collect 1000 SR records for each discussion forum. For unfiltered dataset, we simply collect 1000 SR records without performing similarities to security sources. We have used 1000 records for NSRs.

Security bug dataset:

We have used the security bug report dataset of five projects from three previously published studies [46, 30, 44] as the ground truth for validating our classification models. These projects are: derby, camel, wicket, and ambari from the Ohira et al. [43] dataset. The fifth project is chromium from Peters et al. [46]. We choose these dataset because they are publicly available and have been used in the research community for validating security bug report classification models. Table B.1 provide the properties of the five projects while Table B.2 lists the properties of the actual validation dataset as used in previous studies.

[1] https://cve.mitre.org/
[2] https://cwe.mitre.org/
[3] https://capec.mitre.org/
[4] https://api.stackexchange.com/

Table B.2: Validation dataset

Project	#BRs	#SBRs	SBR(%)
Ambari	500	7	1.4
Wicket	500	6	1.2
Camel	500	18	3.6
Derby	500	42	8.4
Chromium	20970	115	0.5

Table B.1: Validation Project Properties

Project	Domain	Start Date	End Date	#BRs	#SBRs	SBR(%)
Ambari	Hadoop management web UI backed by its RESTful APIs	Sep 26 2011	Aug 8 2014	1000	29	2.9
Wicket	Component-based web application framework for Java programming	Oct 20 2006	Nov 9 2014	1000	10	1.0
Camel	A rule-based routing and mediation engine	Jul 8 2007	Sep 18 2013	1000	32	3.2
Derby	A relational database management system	Sep 28 2004	Sep 17 2014	1000	88	8.8
Chromium	Web browser	Aug 30 2008	Jun 11 2010	41940	192	0.5

Information Retrieval Techniques

We have used the TFIDF [53] metric and word2vec [39] model to identify feature set in our study. We include word embedding because it has been successfully applied to solve software engineering problems where semantic relationships can be inferred [30].

- **Term Frequency-Inverse Document Frequency (TFIDF):** TFIDF combines the term frequency (TF) metric and the inverse document frequency (IDF) metric. Term frequency for a term in a document is computed as the fraction of number of times it occurs in the document to the total words in the document. IDF on the other hand finds the log of the fraction of the total number of documents to the number of documents where a term appears. The intuition behind IDF is that terms that are frequent in all documents may not discriminate very well and will thus be penalized with low IDF. Conversely, terms that occur in a few documents may be more interesting for the documents where they appear and will thus be weighted with higher IDF.

  The TFIDF for term t, in document, d in an entire documents D, corpus can be computed as:

$$\mathbf{TFIDF(t,d)} = \frac{count(t,d)}{\mid d \mid}.log\frac{\mid D \mid}{|\{d \in D : t \in d\}|} \tag{B.1}$$

  By computing the TFIDF for the entire corpus, we retrieve the top-n features with the highest TFIDF.

- **Word Embedding:** is a way of representing a document vocabulary that captures the context of a word in a document, its relation to other words, and its semantic similarity. The basic idea is that if two words share similar contexts, then they will be associated with vectors that are close to each other in the vector space.

  In our approach, we have used the word embedding technique proposed in [39] aka Word2Vec, with an implementation based on the Skip-gram model. This model uses the current word to predict the surrounding context of words in a corpus. The objective function of skip-gram model is to maximize the negative log likelihood of the surrounding context words (w_{i+k}), with a fixed window size m, conditioned on the center word (w_i) over n vocabulary words.

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{-m\leq k\leq m, k\neq 0}^{n} log\, p(w_{i+k}|w_i) \tag{B.2}$$

  The Word2Vec model after training will thus contain a dictionary of words, where each word is associated with a vector representation. We trained a word2vec model for each of CVE, CWE, and CAPEC using a window size of 5, and a dimension of 100. By using the trained Word2Vec model, a document, d (discussion question or security source record) with $w_1, w_2, ..., w_n$ words can use the mean vector representations of the words. If each word, w_i has u_{w_i} vector embedding, the mean vector embedding for d is computed as:

$$u_d = \frac{1}{n}\sum_{i=1}^{n} u_{w_i} \tag{B.3}$$

  We use the mean of the vectors of a discussion or security source record when collecting filtered dataset and we only select n words as features from the vocabulary list for building a machine learning model.

- **Similarity measure:** In this paper, we have used cosine similarity to measure the similarity between two documents. It measures the cosine of the angle between two vectors projected in a multi-dimensional space. Given two documents with vectors A and B, the cosine of the angle between them can be computed as:

$$\textbf{Cosine(A, B)} = \frac{A.B}{\|A\| \times \|B\|} \tag{B.4}$$

  The bigger the cosine value the similar the two documents are to each other. we compare two documents that have been vectorized using TFIDF or Word2Vec model by computing their cosine similarity.

B.3.2 Experiment setup and modelling approach

Table B.3 presents the different combinations of parameters for building our classification models. In total, we constructed 6480 models. We have used six text classification algorithms that have been commonly used in the research environment [46] - Random Forest, Naive Bayes, Support Vector Machine, K-Nearest Neighbor, Logistic Regression, and Multilayer Perceptron Layer. Both

Table B.3: Experiment detail

Models	Total
3 security sources (CVE, CWE, CAPEC) 4 discussion forums (SO, SE, SF, AU) 2 Content categories (Q, Q+A) 5 validation dataset (Ambari,Wicket,Camel,Derby,Chromium) 6 Machine Learning Algorithms 3 Feature types (TFIDF, Word2Vec, Both.) 3 Feature dimensions (100, 200, 300)	6480

the Weka (version 3.8.4) [42] and DeepLearning4J (version 1.0.0.beta6) [14] libraries are used for our experiments. We have used default parameters in the algorithms. As shown in Figure B.2 we perform five experiments for each combination of parameters and compute the mean, maximum, minimum, and standard deviation for the performance metrics. In each experiment, we train a model using 5-fold cross-validation where the training dataset is split into 4 folds (80%) for training and 1 fold (20%) for testing in each round. The model is then validated using each of the security bug validation datasets.

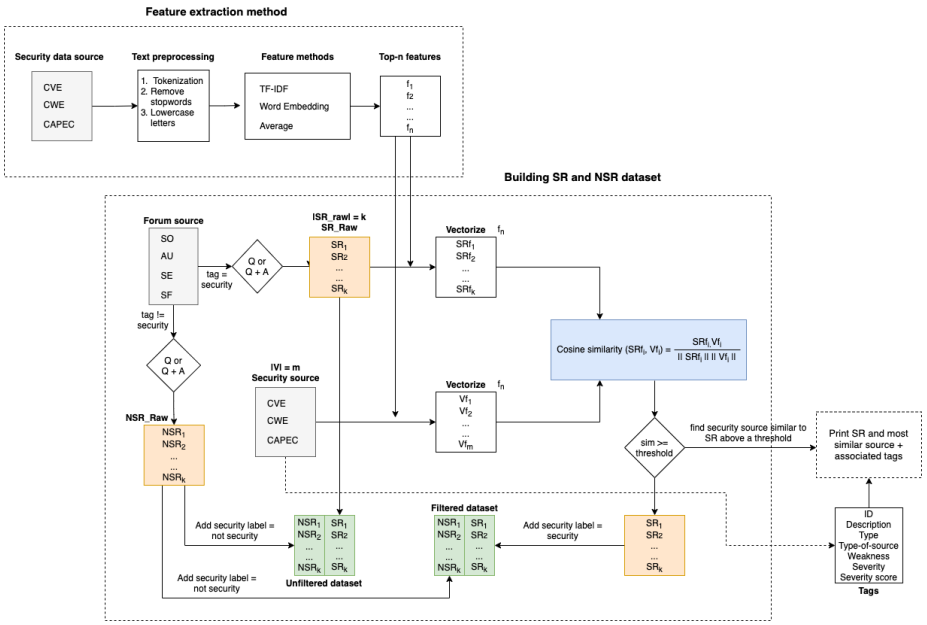

Figure B.1: Dataset collection framework

Performance metrics

We have used recall, precision, probability of false (pf) alarm, f-score, and g-measure as our performance metrics [54]. Both pf and g-measure are used to compare our work with previous studies [46, 30, 44]. These metrics are computed from true positive (TP), true negative (TN), false positive (FP), and false negative (FN) where:

TP = Number of security records correctly identified as security records

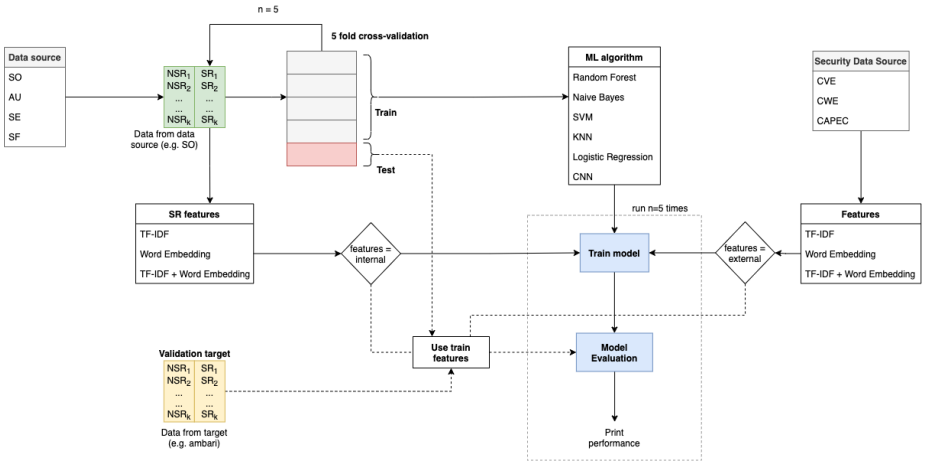

Figure B.2: Experiment framework

TN = Number of non-security records correctly identified as non-security records

FP = Number of non-security records incorrectly identified as security records

FN = Number of security records incorrectly identified as non-security records

Recall = $\frac{TP}{TP+FN}$ **Precision** = $\frac{TP}{TP+FP}$

pf = $\frac{FP}{FP+TN}$

F-Score = $2 * \frac{Precision*Recall}{Precision+Recall}$ **G-measure** = $\frac{2*recall*(100-pf)}{recall+(100-pf)}$

B.3.3 Research questions and analysis

RQ1: What is the performance of vulnerability classification model built with public information sources?

We approach RQ1 in 2 ways:

1. **Comparison to state of the art studies**: We first compare the results of our model to state-of-the-arts studies [46, 30, 44]. We have compared our results to those reported for transfer project prediction since we have used public discussion forums to construct the training dataset for our classification model. We then use the same test dataset as used in all these studies to validate our model. A comparable result to the state-of-the-art studies will be a proof of the usefulness of our approach.

2. **Hypothesis**: We hypothesize that using public security sources to filter training dataset for constructing classification models can produce better classification model. Our null hypothesis (H_0) is thus: Models based on unfiltered dataset by security sources outperform models based on filtered dataset.

To test the hypothesis, we collect the top-50 observations ranked by g-measure for each of the model that use filtered dataset and unfiltered dataset. The data is unpaired, thus we use Wilcoxon signed ranked test [15] (a non-parametric test)

at 95% confidence level to compare the mean of both groups for three metrics - recall, f-score, and g-measure. In addition, we perform effect size check on our results. As noted in Kampenes et al. [31], effect size quantifies the size of the difference between two groups and allows us to judge whether the conclusions drawn from our hypotheses testing are meaningful or not. It might be possible that the effect size is negligible even when the statistical test is significant and vice versa. We apply the Hedges, g standardized effect size measure calculated as: $Hedges, g = \frac{\bar{X}_1 - \bar{X}_2}{S_p}$. Where $\bar{X}_1$ and $\bar{X}_2$ represent the sample means for the classification measures (recall, f-score, and g-measure) and S_p represents the pooled standard deviation computed from the standard deviations of s_1 and s_2 of the two groups. We use the results reported in Software Engineering empirical studies categorized under Table 9 as the basis for comparing our effect sizes. The size category for 284 estimated values for Hedges, g is given as: Small: 0.00-0.376, Medium: 0.378-1.000 and Large: 1.002-3.40. We have used standard statistical packages in R [49].

RQ2: Can we gain additional context for discussion forums from public security sources?

To answer RQ2, we use word2vec model that we have trained on each of the security sources. We then select the first 100 SR records from stackoverflow and serverfault. For each record, we use the mean vector embedding from the word2vec model described in B.3.1 and then perform a cosine similarity to the mean vector embedding of each record from the security source. Using a cosine of 0.5 as our threshold, we collect the top five most similar security sources to a particular forum question. We then provide a qualitative assessments of these mappings to external security sources to identify whether they are meaningful or not. When assessing whether a source reasonably matches a discussion, we use D-Direct, I-Indirect, and N-Not related categories as our classification scheme. Where "Direct" means, the relationship between the two sources are unambigous, "Indirect" means, they can be related (e.g. one provides an example for the other), and "Not related" means, we can not infer any relationship between the two. Two raters have independently evaluated the 100 SR records and compared results in addition to resolving disagreements. A record has one to five suggested matches. We record a one for a category when at least one of the five matches is marked with this category.

B.4 Results & Discussion

B.4.1 RQ1: What is the performance of vulnerability classification model built with public information sources

Comparison to state-of-the-art studies

We compare the best results of the models built with discussion forums to other models reported in the literature. For chromium, the model stands in the second place with 72% gmeasure, 4.2% fscore, and 62.6% recall. For wicket, it ranks second with a g-measure of 88.9%, 10.9% fscore, and 100% recall. The model ranked third for ambari with 84.8% gmeasure, 13% fscore and 85.7% recall. For camel, the model ranked second with 77.4% gmeasure, 19.6% fscore, and

Table B.4: Comparison to state-of-the-art studies

Target	Source	Paper (Model)	Cat	Ratio	Learner	TN	TP	FN	FP	pd	pf	prec	f-score	g-measure
Chromium	Ambari	Peters et al. (clnifarsecsq)	-	-	MLP	19,817	56	59	1,038	48.7	5.0	5.1	9.3	63.9
	Derby	Jiang et al. (rs-selector)	-	1 : 1	MLP	15,346	76	39	5509	66.1	26.4	1.4	2.7	69.6
	Camel	Oyetoyan et al.(fsec-ext+)	TCAI	2.0	LR	16,737	84	31	4109	73.0	19.7	2.0	3.9	76.5
	AU	$CWE-Q-AVG-300-Both(Ext)$	-	-	RF	17,626	72	43	3229	62.6	15.5	2.2	4.2	72.0
Wicket	Camel	Peters et al. (train)	-	-	NB	437	3	3	57	50.0	11.5	5.0	9.1	63.9
	Ambari	Jiang et al. (rs-selector)	-	2 : 1	MLP	396	6	0	98	100.0	19.8	5.8	10.9	88.9
	Camel	Oyetoyan et al.(fsec-ext+)	TCAI	2.0	LR	412	6	0	82	100.0	16.6	6.8	12.8	90.9
	SO	$CAPEC-QA-TFIDF-300-Both(Int)$	-	-	RF	396	6	0	98	100.0	19.8	5.8	10.9	88.9
Ambari	Chromium	Peters et al. (farsecsq)	-	-	MLP	474	3	4	19	42.9	3.9	13.6	20.7	59.3
	Chromium	Jiang et al. (rs-selector)	-	3 : 1	LR	417	7	0	76	100	15.4	8.4	15.6	91.6
	Derby	Oyetoyan et al.(fsec-tfidf+)	-	0.0	SVM	415	6	1	78	85.7	15.8	7.1	13.2	84.9
	SE	$CWE-Q-TFIDF-200-Both(Int)$	-	-	KNN	414	6	1	79	85.7	16.0	7.1	13.0	84.8
Camel	Derby	Peters et al. (farsectwo)	-	-	NB	371	8	10	110	44.4	22.9	6.8	11.8	56.4
	Derby	Jiang et al. (rs-selector)	-	9 : 1	LR	425	13	5	57	72.2	11.8	18.6	29.6	79.4
	Derby	Oyetoyan et al.(fsec-ext+)	TCAI	1.0	LR	359	13	5	123	72.2	25.5	9.6	16.9	73.3
	AU	$CAPEC+Q+Word2Vec-200-Both(Ext)$	-	-	KNN	371	14	4	111	77.8	23.0	11.2	19.6	77.4
Derby	Chromium	Peters et al. (clnifarsecsq)	-	-	NB	372	19	23	86	45.2	18.8	18.1	25.9	58.1
	Ambari	Jiang et al. (ms-selector)	-	2 : 1	MLP	295	30	12	164	71.4	35.7	15.5	25.4	67.7
	Wicket	Oyetoyan et al.(fsec-ext+)	C	2.0	LR	419	27	15	39	64.3	8.5	40.9	50.0	75.5
	SE	$CWE-QA-Word2Vec-300-TFIDF(Ext)$	-	-	NB	360	30	12	98	71.4	21.4	23.4	35.3	74.8

Table B.5: Summary of results of filtered dataset with the best average g-measure

								pd		pf		prec		fscore		g-measure	
Target	Forum	Content	Feature.Method	Feature.source	Feature.dim	Feature.Learner	Learner	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev
Chromium	SO	Q	Word2Vec	CVE	200	Both(Ext)	LR	52.1	1.0	8.5	2.6	1.9	0.5	3.7	0.9	66.4	0.4
	SF	Q	TFIDF	CWE	300	Both(Ext)	RF	53.9	1.3	11.5	1.6	1.9	0.2	3.7	0.4	66.6	0.6
	SO	Q	AVG	CAPEC	200	Word2Vec(Ext)	LR	52.2	0.9	8.9	1.5	2.3	0.4	4.4	0.6	66.4	0.6
Wicket	AU	Q	Word2Vec	CWE	300	TFIDF(Int)	RF	83.3	0.0	9.3	2.2	6.0	1.3	11.2	2.1	83.8	1.1
	AU	QA	TFIDF	CAPEC	300	Word2Vec(Int)	NB	66.7	0.0	5.9	1.2	7.8	1.4	14.0	2.1	76.8	0.4
	SO	Q	AVG	CWE	100	Both(Ext)	NB	66.7	0.0	13.6	2.0	3.9	0.6	7.5	1.0	72.9	0.8
Ambari	AU	QA	Word2Vec	CWE	300	TFIDF(Ext)	RF	71.4	0.0	10.3	0.9	7.1	0.6	12.9	0.9	78.4	0.4
	SO	QA	TFIDF	CAPEC	100	TFIDF(Ext)	RF	71.4	0.0	10.7	0.9	7.1	0.5	12.9	0.9	78.4	0.4
	SE	Q	AVG	CWE	100	Word2Vec(Ext)	RF	71.4	0.0	8.1	0.7	8.9	0.7	15.9	1.1	79.5	0.3
Camel	AU	Q	Word2Vec	CAPEC	200	Both(Ext)	RF	72.2	2.2	23.0	1.4	9.1	0.8	16.1	1.3	72.6	1.8
	AU	Q	TFIDF	CAPEC	200	TFIDF(Ext)	RF	72.2	2.2	27.6	2.4	7.3	0.5	13.3	0.8	68.8	1.2
	AU	Q	AVG	CAPEC	200	Word2Vec(Ext)	RF	66.7	2.2	23.2	1.4	8.6	0.4	15.2	0.6	69.9	1.0
Derby	AU	Q	Word2Vec	CWE	200	TFIDF(Ext)	SVM	59.5	1.8	23.6	0.7	18.2	0.6	28.1	0.9	66.9	1.1
	AU	Q	TFIDF	CWE	200	TFIDF(Ext)	NB	66.7	3.6	25.1	5.2	14.1	1.9	23.5	2.3	65.2	1.8
	AU	Q	AVG	CWE	300	TFIDF(Ext)	SVM	52.4	4.6	17.2	6.2	15.5	2.7	24.8	2.6	64.2	0.8

Table B.6: Summary of results of unfiltered dataset with the best average g-measure

							pd		pf		prec		fscore		g-measure	
Target	Forum	Content	Feature.source	Feature.dim	Feature-Learner	Learner	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev	Mean	Std.dev
Chromium	SE	Q	CVE	200	Word2Vec(Ext)	LR	54,8	0,9	13,2	1,3	1,9	0,2	3,7	0,3	67,1	0,5
	SF	Q	CWE	300	TFIDF(Ext)	LR	63,5	1,7	29,5	0,9	1,2	0,0	2,3	0,1	66,6	0,7
	SE	Q	CVE	200	Both(Ext)	LR	50,4	1,5	6,1	2,4	2,3	0,8	4,4	1,4	65,6	1,1
Wicket	SO	QA	-	200	Word2Vec(Int)	LR	66.7	6.7	9.7	2.1	6.2	1.3	11.3	2.1	75.7	3.7
	SE	Q	-	200	TFIDF(Int)	NB	66.7	0.0	8.9	0.6	7.0	0.5	12.7	0.7	76.3	0.2
	SE	Q	-	200	Both(Int)	NB	66.7	0.0	8.5	1.3	6.2	0.9	11.3	1.4	75.7	0.5
Ambari	AU	Q	CWE	100	Word2Vec(Ext)	SVM	71.4	0.0	12.8	0.7	6.4	0.3	11.7	0.5	77.7	0.3
	AU	QA	-	300	TFIDF(Int)	RF	71.4	0.0	16.0	1.7	4.8	0.5	9.1	0.8	75.5	0.7
	AU	QA	CWE	100	Both(Ext)	LR	71.4	0.0	13.2	2.1	5.1	0.7	9.4	1.2	75.9	0.9
Camel	AU	Q	CAPEC	200	Word2Vec(Ext)	RF	61.1	6.5	21.2	2.7	8.6	0.4	15.2	0.6	68.9	2.3
	AU	Q	CAPEC	200	TFIDF(Ext)	RF	61.1	5.4	28.2	0.9	7.4	0.5	13.2	0.9	65.8	2.6
	AU	Q	CAPEC	300	Both(Ext)	RF	61.1	8.2	26.9	1.6	7.8	0.5	13.8	0.9	66.5	3.2
Derby	SO	Q	CVE	100	Word2Vec(Ext)	LR	57.1	2.6	21.8	1.3	18.2	1.2	27.7	1.6	65.5	1.8
	AU	Q	CWE	200	TFIDF(Ext)	NB	61.9	1.8	28.4	2.2	15.1	0.6	24.7	0.6	66.2	0.3
	SE	QA	CWE	200	Both(Ext)	NB	54.8	8.4	17.5	9.5	13.9	3.1	23.6	3.0	65.2	0.9

best recall of 77.8%. Lastly, for derby, the model also ranked second with 74.8% gmeasure, 35.3% fscore, and 71.4% fscore. Clearly, the results are competitive to existing results. We can infer that the public discussion forum contains textual reports that can be semantically similar and useful to build security bug report classification models. Furthermore, it can be possible to use the model to tag bug reports. As an example, a bug report in chromium can be tagged with related CWE and CAPEC from our word2vec model as follows:

Bug: *Chrome Buffer Overlow Vulnerability - "SaveAs" Function s...@bkav.com.vn SVRT - Bkis have just discovered vulnerability in Google Chrome 0.2.149.27 and would like to inform you with this. Here comes the report Details - Type of Issue Buffer Overflow. - Affected Software Google Chrome 0.2.149.27. - Exploitation Environment Google Chrome (Language Vietnamese) on Windows XP SP2. - Impact Remote code execution - Description The vulnerability is caused due to a boundary error when handling the "SaveAs" function. On saving a malicious page with an overly long title (¡title¿ tag in HTML) the program causes a stack-based overflow and makes it possible for attackers to execute arbitrary code*
CWE-130: *If an attacker can manipulate the length parameter associated with an input such that it is inconsistent with the actual length of the input this can be leveraged to cause the target application to behave in unexpected and possibly malicious ways One of the possible motives for doing so is to pass in arbitrarily large input to the application Another possible motivation is the modification of application state by including invalid data for subsequent properties of the application Such weaknesses commonly lead to attacks such as buffer overflows and execution of arbitrary code*
CAPEC-242: *code injection "An adversary exploits a weakness in input validation on the target to inject new code into that which is currently executing. This differs from code inclusion in that code inclusion involves the addition or replacement of a reference to a code file*

Comparing models with filtered dataset to models with unfiltered dataset

We report the summary statistics of the model with the best g-measure average for both the filtered dataset in Table B.5 and unfiltered dataset in Table B.6. In Table B.5, the results show the discussion forum where the training dataset is collected, whether we use question (Q), or a combination of question and answers (QA), the method used to collect features from the dataset (Feature.Method), the security data source of the features (Feature.source), the size of the feature (Feature.dim), the feature used as dictionary list for the leaner algorithm (Feature.Learner), this can be either external or internal, and the learner algorithm (Learner). The metrics for chromium across the three methods are similar - recalls between 52% and 53%, gmeasures between 66.4% and 66.6%, and precisions between 1.9% and 2.3%. These patterns are the same for ambari, camel, and derby. Only wicket has a clear best model with 83% recall and gmeasure. CAPEC and CWE are dominant security sources and askubuntu (AU) featured most. Both CAPEC and CWE relate to attack scenarios and weaknesses which are common in software projects unlike the CVE that describe real vulnerabilities in specific products. We cannot explain the reason for AU dominating the dataset. External features from security sources appear to produce the best and stable models across the projects.

In comparison to unfiltered dataset (Table B.6), the performance metrics are similar mostly across the projects. The exception is wicket with a clear best model recorded with filtered dataset. This is confirmed by the hypothesis test for

Table B.7: Hypothesis: filtered dataset vs. unfiltered dataset

		Word2Vec		TFIDF	
Project	Metric	p-value	Hedges,g	p-value	Hedges,g
Chromium	recall (pd)	0.99	−1.28	0.94	−0.60
	f-score	< 0.001	1.17	0.93	−0.41
	g-measure	0.99	−1.11	0.84	−0.19
Wicket	recall (pd)	< 0.0001	1.02	0.31	0.11
	f-score	0.021	0.47	0.10	0.18
	g-measure	< 0.0001	1.97	0.006	0.49
Ambari	recall (pd)	0.99	−0.64	0.99	−0.48
	f-score	< 0.0001	1.82	< 0.0001	0.75
	g-measure	< 0.0001	1.67	< 0.0001	0.71
Camel	recall (pd)	0.98	−0.43	0.88	$-0.19s$
	f-score	< 0.0001	0.60	< 0.0001	0.40
	g-measure	0.01	0.42	< 0.0001	0.89
Derby	recall (pd)	0.80	−0.18	0.99	−0.56
	f-score	< 0.0001	0.47	0.88	−0.49
	g-measure	< 0.0001	0.71	0.99	−0.91

Table B.8: Percentage of CWE, CAPEC, and CVE matches in StackOverflow and ServerFault discussion questions

Forum	Category	CWE	CAPEC	CVE
StackOverflow	Direct	12	10	0
	Indirect	44	54	14
ServerFault	Direct	14	8	0
	Indirect	20	14	18

word2vec model in Table B.7 where the test for the metrics (recall, fscore, and gmeasure) are significant for filtered dataset with effect sizes varying between medium (0.47) and large (1.97). Another benefit of filtered dataset could be seen in the significant test under word2vec model of fscore (harmonic mean between recall and precision) and g-measure (harmonic mean between recall and true negative rate). Both the fscores and gmeasures for wicket, ambari, camel, derby and chromium (only fscore) are statistically significant with effect sizes between medium and large. The recall however is better with the unfiltered dataset. Another observation from Table B.6 is that using externally generated features from security sources produce the best models across the projects with only wicket as an exception. We can infer that extracting features from security sources can be beneficial for security report classification models.

B.4.2 RQ2: Can we gain additional context for discussion forums from public security sources?

We report the result of our qualitative assessments of the matches between the security sources and discussion forum questions.

Table B.8 presents the percentage of matching records from qualitative assessment of 100 security-related questions in both stackoverflow and serverfault. In summary, for stackoverflow, 12% of CWE has direct matches while 44% has indirect matches. For CAPEC, 10% has direct matches and 54% has indirect matches. For CVE, 14% has indirect matches while there was no direct match. For serverfault, 14% of CWE has direct matches while 20% has indirect matches. For CAPEC, 8% has direct matches and 14% has indirect matches. For CVE, 18% has indirect matches while there was no direct match. The empty match for

CVE in the sample of SR we assessed can be explained first based on the small size of CVE records used for training the word2vec model and second, because the discussion forum questions we assessed are not attack incidents. Tables B.9 and B.10 present specific examples of questions with direct and indirect matches to the security sources. The matches provide, in some cases examples of incident (CVE) that can be relevant to the question. For example, the user with question-Id 934731 is looking to log all account management activities for at least 6 months. However, CVE-2020-15095 in Table B.10 presents real world information exposure vulnerability in npm cli through log files that can help the user think through certain mitigation while implementing this feature. CAPEC-93: Log Injection-Tampering-Forging, further amplifies the pattern of attack. Thus, the stakeholder can think of mitigation in advance such as digitally signing every record in the log file in order to detect tampering.

In some other cases, the matches only provide information to users. For examples, the questions id - 1057084 (fixing weak TLS), 104870 (Memory Pressure Protection feature) in Table B.10 and 63605452 (XML injection), 28606689 (Screen Capture in Android), and 332365 (SQL injection) all have CWE and CAPEC matches that describe and discuss the implications of the security questions.

B.4.3 Discussion

Using the combination of the classification model and word2vec model, the results demonstrate a possibility to both identify false positives and false negatives in the classification model. Figure 3 shows a possible way to combine the models we have developed in this study for the purpose of refining classification model. The idea is to use the word2vec model as a validation model, where the classification result (SR or NSR) is passed through the word2vec model. Using a certain threshold, we can check whether additional context from CWE, CAPEC, or CVE can be derived. If a classification result is NSR but has a context from the word2vec model, we can conclude that it is a SR (FN -¿ TP). Similarly, if a classification result is SR and does not have a context, then this may probably be a NSR (FP -¿ TN). However, such has to be re-inspected. In addition, it can also be that results from the word2vec model can be used during training for re-training models. Lastly, we identify that security report classification model constructed using discussion forum dataset has competitive results with models developed using bug reports. One advantage of using discussion forum dataset is the availability of security training datasets which is a challenge with bug reports [49]. As a result, there is possibility for improving model based only on dataset.

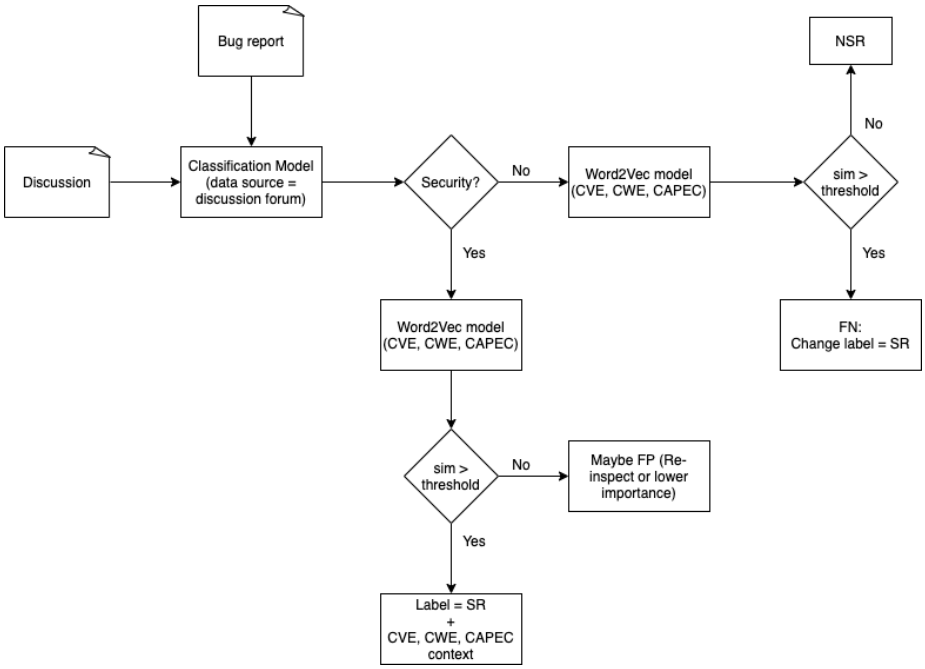

Figure B.3: Using a combination of classification model and word-embedding model for validation

B.5 Related study

Our study intersects between public information security sources, bug report classification, and linking knowledge units. In this section, we provide a review of studies in these areas that are related to our work.

B.5.1 Linking Knowledge Units

Vasilescu et al. [58] investigates the relationship between StackOverflow activities and code changes in GitHub. They find that GitHub committers ask fewer questions and provide more answers than others and that the StackOverflow activity rate correlates with the code changing activity in GitHub.

Ponzanelli et al. [48] proposes a novel approach that takes a context in the IDE and automatically provides recommendation of discussion retrieved from StackOverflow that can help developers

Xu et al. [66] formulates the problem of predicting semantically linkable knowledge units as a multiclass classification problem, and solve the problem using deep learning techniques.

Xu et al. [65] proposes a query-focused multi-answer-posts summarization task for a given technical question, AnswerBot. Their user study results showed that answers generated by their approach were relevant, useful and diverse.

Table B.9: Examples of CWE, CAPEC, and CVE matches to StackOverflow discussion questions

Q-ID	Question	CWE	CAPEC	CVE
63605452	How to prevent XML injection - I got a vulnerability report. XML is injected in the URL "XInclude". I'm trying to find a validation to prevent the XML to be executed. My web application is built using Visual Studio C# with webforms. I was thinking to validate this from the web.config or IIS. I'm not sure if I have to add code to validate or parse the XML...	CWE-611: Improper Restriction of XML External Entity Reference - The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output...	CAPEC-250: An attacker utilizes crafted XML user-controllable input to probe, attack, and inject data into the XML database, using techniques similar to SQL injection. The user-controllable input can allow for unauthorized viewing of data, bypassing authentication or the front-end application for direct XML database access, and possibly altering database information	
28606689	How to prevent Screen Capture in Android - Is it possible to prevent the screen recording in Android Application? I would like to develop an Android Secure Application. In that I need to detect screen recording software which are running background and kill them. I have used SECURE FLAG for prevent screenshots. But I dont know is it possible to prevent Video capturing of Android Screen also. Let me know how to prevent screen capturing (video / screenshots)...		CAPEC-648: Collect Data from Screen Capture - An adversary gathers sensitive information by exploiting the system's screen capture functionality. Through screenshots, the adversary aims to see what happens on the screen over the course of an operation. The adversary can leverage information gathered in order to carry out further attacks...	
332365	How does the SQL injection from the "Bobby Tables" XKCD comic work? What does this SQL do: Robert'); DROP TABLE STUDENTS; – I know both ' and – are for comments, but doesn't the word DROP get commented as well since it is part of the same line?...	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component...		

Table B.10: Examples of CWE, CAPEC, and CVE matches to ServerFault discussion questions

Q-ID	Question	CWE	CAPEC	CVE
934731	Using auditd and retaining log files for 6 months. - Find a way to log all account management activities (e.g., account creation, modification, deletion, etc.) on an Ubuntu 16.04 LTS server and retain the logging information for at least 6 months...		CAPEC-93: Log Injection-Tampering-Forging - This attack targets the log files of the target host. The attacker injects, manipulates or forges malicious log entries in the log file, allowing them to mislead a log audit, cover traces of attack, or perform other malicious actions. The target host is not properly controlling log access. As a result tainted data is resulting in the log files leading to a failure in accountability, non-repudiation and incident forensics capability.	CVE-2020-15095: versions of the npm cli prior to are vulnerable to an information exposure vulnerability through log files the cli supports urls like protocol hostname path the password value is not redacted and is printed to stdout and also to any generated log files
1057084	How to fix Weak TLS 1.2 Encryption - I have a requirement to disable below weak TLS ciphers in Windows Server 2016. I tried to reasearch and it says "The Microsoft SCHANNEL team does not support directly manipulating the Group Policy and Default Cipher suite locations in the registry" Please advise. Thank you in advance...	CWE-326: Inadequate Encryption Strength - A weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of succeeding using current attack methods and resources.		
104870	Memory Pressure Protection Feature for TCP Stack - Provided by Microsoft Security Update KB967723 - We've been having a lot of funky issues with some of our web based applications that allow clients to submit lot of image files to our servers. Lots of ports are used in the process... There doesn't appear to be a pattern and sometimes it works and other times is doesn't. Typically we've noticed it when server is under load.I'm curious what others think about this MPP and any issues that you may have experienced from it...	CWE-406: Insufficient Control of Network Message Volume (Network Amplification) - In the absence of a policy to restrict asymmetric resource consumption, the application or system cannot distinguish between legitimate transmissions and traffic intended to serve as an amplifying attack on target systems. Systems can often be configured to restrict the amount of traffic sent out on behalf of a client, based on the client's origin or access level...		

Bogdanova et al. [6] aims to detect semantically equivalent questions in online user forums by performing an extensive number of experiment using data from two different Stack Exchange forums, and compare standard machine learning methods with a convolutional neural network. Their results showed that the convolutional neural network with in-domain word embeddings achieved high performance even with limtied training data.

Zhang et al. [68] proposes an automated approach to duplicate question detection that takes a new question as input and detects potential duplicates of this question considering multiple factors. Their results show that the approach improves the recall-rate of 40.63% compared to the standard search engine of Stack Overflow.

Yang et al. [67] proposes a novel approach for similar bug recommendation using traditional information retrieval techniques and a word embedding technique. The results show that their approach improves the performance of similar bug recommendation system, NextBug, significantly and substantially.

Li et al. [36] conducts an empirical study to explore the characteristics of a large number of links within popular Python projects in GitHub.

Cai et al. [8] proposes a tool, AnswerBot, which enables to automatically generate an answer summary for a technical problem based on Stack Overflow. The results showed thath the answer summaries generated by AnswerBot were more relevant, useful and diverse than Google and Stack Overflow search engine.

Ruohonen [50] investigates the use of common textual information retrieval techniques to map reported vulnerability to their corresponding software weaknesses.Their results show that explicit referencing of vulnerability and weaknesses yield more consistent results compared to information retrieval techniques.

B.5.2 Public Information Security Sources

Sauerwein et al. [51] conducts a triangulation study to identify and analyze public information security data sources, as well as introducing a taxonomy to classify and compare these data sources. Their investigations showed that research and practice rely on a large variety of heterogeneous information security data sources, which makes it difficult to integrate and use for information security and risk management processes.

Felderer et al. [16] proposes a framework for security data extraction, processing and application. The framework consists of a security data collection component and an analysis component, as well as a security knowledge generation component.

Wijayasekara et al. [61] discusses existing bug data mining classifiers, and presents an analysis of vulnerability databases showing the necessity to mine common publicly available bug databases for hidden impact vulnerabilities.

Pletea et al. [47] evaluates the presence and atmosphere surrounding security-related discussions on GitHub from discussions around commits and pull requests.

Ohira et al. [43] introduces a dataset of high impact bugs and with security implications that was created by manually reviewing four thousand issue reports in four open source projects.

Islam et al. [29] studies 2716 high-quality posts from Stack Overflow and 500 bug fix commits from Github about five popular deep learning libraries to understand the types of bugs, root causes of bugs, impacts of bugs, bug-prone stage of deep learning pipeline as well as whether there are some common antipatterns found in this buggy software.

Wei et al. [59] proposes a semantics-aware approach for warning prioritization in Link, a static analyzer for detecting bugs/issues in Android apps, called OASIS. OASIS combines program analysis and NLP techniques to recover intrinsic links between the Link warnings and the user complaints of an app.

B.5.3 Bug Report Classification

Wu et al. [64] proposes an automated data labeling approach based on iterative voting classification for identifying security bug reports.

Tyo [57] explores the distribution and characteristics of security vulnerabilities in bug tracking systems, as well as data analytics approaches for automatic classification of security bug reports. They used supervised learning algorithms as well as a novel unsupervised machine learning approach. Their results showed most consistently good results from the learning algorithm Naive Bayes across all datasets used.

Peters et al. [46] proposes a framework, FARSEC, for filtering and ranking bug report for reducing the presence of security bug reports that removes non-security bug reports containing security related keywords before building a prediction model. They demonstrate that FARSEC improves the performance of text-based prediction models for security bug reports in 90% of cases.

Gegick et al. [23] develops an approach to identify mislabeled security bug reports that applies text mining on natural-language descriptions of bug reports to train a statistical model.

Shu et al. [56] aims to aid software developers to better classify bug reports that identify security vulnerabilities as security bug reports through parameter tuning of learners and data pre-processor.

Goseva-Popstojanova et al. [25] focuses on automated classification of software bug reports to security and not-security related using both supervised and unsupervised approaches. Their results showed that the supervised learning slightly outperforms the unsupervised learning, at the expense of labeling the training set. In general, the results showed that datasets with more security information lead to better performance.

Wijayasekara et al. [60] presents a hidden impact bug identification methodology by means of text mining bug databases. Wright et al. [63] conducts an experiment using the MySQL bug report database to estimate the number of misclassified bugs yet to be identified as vulnerabilities.

Gantzer [22] focuses on exploring techniques that have potential to improve the

performance of automated classification of software bug reports as security or non-security related. They use feature selection, clustering and deep learning for classification.

Shu et al. [55] proposes a method, SWIFT, that combines learner hyperparameter optimization and pre-processor hyperparameter optimization for distinguishing security-related bug reports in a product's bug database. Their results show that their approach achieves better performance in a fast way than existing state-of-the-art method.

Jiang et al. [30] proposes a novel approach, LTRWES, that incorporates learning to rank a word embedding into the identification of SBRs. Their results show that the proposed method outperforms state-of-the-art method.

Zhou et al. [69] describes an efficient automatic vulnerability identification system for tracking large-scale projects in real time using natural language processing and machine learning techniques.

Kudjo et al. [34] introduces a new approach for vulnerability classification using term frequency and inverse gravity moment (TF-IGM). Their results show that TF-IGM outperforms the benchmark method across the applications used in the study.

Chawla et al. [9] presents an automated technique for bug labeling using TF-IDF and LSI. Their results showed better results from using LSI along with TF-IDF compared to using TF-IDF alone.

Das et al. [13] proposes a learning based approach to identify security and performance bug reports addressing class-bias and feature-skew phenomenon.

Mostafa et al. [41] proposes an automatic approach to identify security bug reports in open bug repositories using semi-supervised learning and keyword-based pre-filtering based on keywords mined from existing security related textual description in the CVE. Their results show that the approach outperforms the best baseline approaches.

Scandariato et al. [52] presents an approach based on machine learning to predict which components of a software application contain security vulnerabilities by text mining the source code of the components.

Oyetoyan et al. [44] investigates whether a generic text classification model can be developed for classifying security related messages in software development project communications. They used harvested features classified into four categories to build classification models. Their approach outperform a state-of-the-art prediction model for security bug reports.

Our study is different from the above studies as we have combined these three sub-fields in order to contextualize security report classifications models.

B.6 Threats to validity

Data sampling: We have collected a subset of the security related records in the discussion forums. These samples do not represent all types of dataset in the forums. E.g. StackOverflow contains over 80000 SR records. We have only used

2000 records. Using a bigger sample size can lead to different results. However, our validation results are proof that the sample size is a good representation.

Modelling: There are several parameters that can be tuned in the algorithms we have used. We have used the default settings for the parameters of the algorithms during model construction. In the cases where we have used specific parameters, we have reported them in the experiment section. It is possible that tuning the parameters might lead to different results.

Generalization: The observations we have made are based on four discussion forums on stackexchange and five open source bug report dataset. We cannot claim that dataset from other forums and projects will produce the same results. Therefore, further studies will be necessary to generalize the results across other forums and software project dataset.

B.7 Conclusion

We have investigated using public security sources such as CWE, CAPEC, and CVE to provide additional context for security discussions on stackexchange forums. We argue that such context can further clarify questions and provide examples of weaknesses, attack patterns, and real-world scenarios related to the question.

We built models by using forums' dataset that we have filtered with security sources and compare their performances to unfiltered dataset from the forum. By validating the models on five popular security bug report dataset, we establish that features extracted from security sources can be useful to build security report models. We also find that filtered dataset does not produce models with higher recalls to unfiltered ones, however, they produce models that are better in fscore and gmeasure. Lastly, our models' performances are competitive to state-of-the-art studies.

We assess the CWE, CAPEC, and CVE recommendations from our word2vec models for a sample of stackoverflow and serverfault records. We find 12% - 14% direct matches for CWE, 8% - 10% direct matches for CAPEC, 20% - 44% indirect matches for CWE, 14% - 54% indirect matches for CAPEC, and 14% - 18% indirect matches for CVE.

In conclusion, our results demonstrate a useful layer of validation for security classification models, by providing additional security context information.

Bibliography

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. How internet resources might be helping you develop faster but less securely. *IEEE Security & Privacy*, 15(2):50–60, 2017.

[2] M. Allahyari, Seyed Amin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and K. Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *ArXiv*, abs/1707.02919, 2017.

[3] Quality Assurance. Software quality attributes, 2011.

[4] Wei Bai, Omer Akgul, and Michelle L Mazurek. A qualitative investigation of insecure code propagation from online forums. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 34–48. IEEE, 2019.

[5] Mario Barbacci, Mark H Klein, Thomas A Longstaff, and Charles B Weinstock. Quality attributes. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1995.

[6] D. Bogdanova, C. D. Santos, L. Barbosa, and Bianca Zadrozny. Detecting semantically equivalent questions in online user forums. In *CoNLL*, 2015.

[7] J. Brownlee. *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery, 2017.

[8] L. Cai, Haoye Wang, Bowen Xu, Q. Huang, Xin Xia, D. Lo, and Zhenchang Xing. Answerbot: an answer summary generation tool based on stack overflow. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[9] Indu Chawla and S. Singh. Automatic bug labeling using semantic information from lsi. *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 376–381, 2014.

[10] The MITRE Corporation. About capec, 2019.

[11] The MITRE Corporation. About cve, 2021.

[12] The MITRE Corporation. About cwe, 2021.

[13] Dipok Chandra Das and Md. Rayhanur Rahman. Security and performance bug reports identification with class-imbalance sampling and feature selection. *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pages 316–321, 2018.

[14] Deeplearning4j. Eclipse deeplearning4j, 2020.

[15] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[16] M. Felderer and Irdin Pekaric. Research challenges in empowering agile teams with security knowledge based on public and private information sources. In *SecSE@ESORICS*, 2017.

[17] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136. IEEE, 2017.

[18] Interaction Design Foundation. Usability, 2021.

[19] The Apache Software Foundation. Welcome to apache maven, 2021.

[20] Wei Fu and Tim Menzies. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pages 49–60, 2017.

[21] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.

[22] Tanner D Gantzer. Security bug report classification using feature selection, clustering, and deep learning. 2019.

[23] M. Gegick, Pete Rotella, and T. Xie. Identifying security bug reports via text mining: An industrial case study. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 11–20, 2010.

[24] Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[25] K. Goseva-Popstojanova and Jacob Tyo. Identification of security related bug reports via text mining using supervised and unsupervised classification. *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 344–355, 2018.

[26] Zellig Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[27] GitHub Inc. Where the world builds software, 2021.

[28] Stack Exchange Inc. About us, 2021.

[29] M. J. Islam, Giang Nguyen, Rangeet Pan, and H. Rajan. A comprehensive study on deep learning bug characteristics. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[30] Yuan Jiang, Pengcheng Lu, Xiaohong Su, and Tiantian Wang. Ltrwes: A new framework for security bug report detection. *Inf. Softw. Technol.*, 124:106314, 2020.

[31] Vigdis By Kampenes, Tore Dybå, Jo E Hannay, and Dag IK Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, 2007.

[32] Tom Kenter and M. Rijke. Short text similarity with word embeddings. In *CIKM '15*, 2015.

[33] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[34] P. Kudjo, J. Chen, Minmin Zhou, Solomon Mensah, and Rubing Huang. Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment. *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 248–259, 2019.

[35] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[36] L. Li, Z. Ren, Xiaochen Li, W. Zou, and He Jiang. How are issue units linked? empirical study on the linking behavior in github. *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 386–395, 2018.

[37] M. Lopez and J. Kalita. Deep learning applied to nlp. *ArXiv*, abs/1703.03091, 2017.

[38] QualCode Ltd. Quality attributes, 2021.

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[41] Shaikh Mostafa and Xiaoyin Wang. Automatic identification of security bug reports via semi-supervised learning and cve mining. 2020.

[42] University of Waikato. Weka, 2021.

[43] M. Ohira, Yutaro Kashiwa, Yosuke Yamatani, Hayato Yoshiyuki, Yoshiya Maeda, Nachai Limsettho, K. Fujino, Hideaki Hata, Akinori Ihara, and K. Matsumoto. A dataset of high impact bugs: Manually-classified issue reports. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 518–521, 2015.

[44] T.D. Oyetoyan and P. Morrison. An Improved Text Classification Modelling Approach to Identify Security Messages in Heterogeneous Projects. *Software Quality Journal*, in press.

[45] Tosin Daniel Oyetoyan, Bisera Milosheska, Mari Grini, and Daniela Soares Cruzes. Myths and facts about static application security testing tools: An action research at telenor digital. In Juan Garbajosa, Xiaofeng Wang,

and Ademar Aguiar, editors, *Agile Processes in Software Engineering and Extreme Programming - 19th International Conference, XP 2018, Porto, Portugal, May 21-25, 2018, Proceedings*, volume 314 of *Lecture Notes in Business Information Processing*, pages 86–103. Springer, 2018.

[46] F. Peters, T. Tun, Yijun Yu, and B. Nuseibeh. Text filtering and ranking for security bug report prediction. *IEEE Transactions on Software Engineering*, 45:615–631, 2019.

[47] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In *MSR 2014*, 2014.

[48] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 102–111, 2014.

[49] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[50] Jukka Ruohonen and Ville Leppänen. Toward validation of textual information retrieval techniques for software weaknesses. In *International Conference on Database and Expert Systems Applications*, pages 265–277. Springer, 2018.

[51] Clemens Sauerwein, Irdin Pekaric, M. Felderer, and R. Breu. An analysis and classification of public information security data sources used in research and practice. *Comput. Secur.*, 82:140–155, 2019.

[52] R. Scandariato, James Walden, A. Hovsepyan, and W. Joosen. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40:993–1006, 2014.

[53] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008.

[54] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34:1–47, 2002.

[55] Rui Shu, Tianpei Xia, Jianfeng Chen, Laurie Williams, and T. Menzies. Improved recognition of security bugs via dual hyperparameter optimization. *ArXiv*, abs/1911.02476, 2019.

[56] Rui Shu, Tianpei Xia, Laurie Williams, and T. Menzies. Better security bug report classification via hyperparameter optimization. *ArXiv*, abs/1905.06872, 2019.

[57] Jacob P. Tyo. Empirical analysis and automated classification of security bug reports. 2016.

[58] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: Associations between software development and crowd-

sourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195. IEEE, 2013.

[59] L. Wei, Yepang Liu, and S. Cheung. Oasis: prioritizing static analysis warnings for android apps based on app user reviews. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017.

[60] D. Wijayasekara, M. Manic, and M. McQueen. Vulnerability identification and classification via text mining bug databases. *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pages 3612–3618, 2014.

[61] D. Wijayasekara, M. Manic, J. L. Wright, and M. McQueen. Mining bug databases for unidentified software vulnerabilities. *2012 5th International Conference on Human System Interactions*, pages 89–96, 2012.

[62] C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *EASE '14*, 2014.

[63] J. L. Wright, Jason W. Larsen, and M. McQueen. Estimating software vulnerabilities: A case study based on the misclassification of bugs in mysql server. *2013 International Conference on Availability, Reliability and Security*, pages 72–81, 2013.

[64] X. Wu, Wei Zheng, Xiang Chen, F. Wang, and D. Mu. Cve-assisted large-scale security bug report dataset construction method. *J. Syst. Softw.*, 160, 2020.

[65] Bowen Xu, Zhenchang Xing, Xin Xia, and D. Lo. Answerbot: Automated generation of answer summary to developers' technical questions. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 706–716, 2017.

[66] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, G. Chen, and S. Li. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 51–62, 2016.

[67] Xinli Yang, D. Lo, Xin Xia, L. Bao, and Jianling Sun. Combining word embedding with information retrieval to recommend similar bug reports. *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 127–137, 2016.

[68] Y. Zhang, D. Lo, Xin Xia, and Jianling Sun. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology*, 30:981–997, 2015.

[69] Y. Zhou and Asankhaya Sharma. Automated identification of security issues from commit messages and bug reports. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017.

[70] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 421–428. IEEE, 2010.