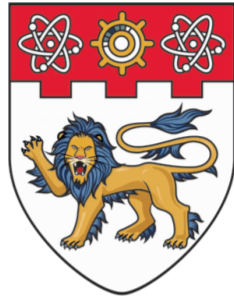


NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

Automated Vulnerability Detection System Based on Commit Messages

Wan Liuyang

School of Computer Science and Engineering

A thesis submitted in Partial Fulfilment of the Requirements for the
Degree of Master of Engineering of the
Nanyang Technological University

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

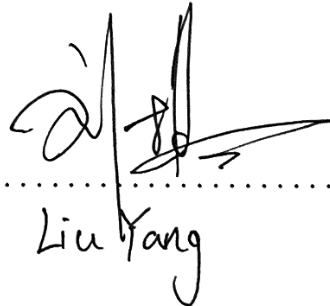
05/03/19 ……………………
Date

万浏洋 ……………………
Wan Liuyang

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

05/03/19
....
Date



Liu Yang

Authorship Attribution Statement

This thesis **does not** contain any materials from papers published in peer-reviewed journals or from papers accepted at conferences in which I am listed as an author.

05/03/19
Date

万浏洋
Wan Liuyang

Abstract

Vulnerabilities in Open Source Software (OSS) are the major culprits of cyber-attacks and security breaches today. To avoid repetitive development and speed up release cycle, software teams nowadays are increasingly relying on OSS. However, many OSS users are unware of the vulnerable components they are using. The de-facto security advisory standard, National Vulnerability Database (NVD) is known to suffer from poor coverage and inconsistency. Sometimes it will take weeks or even months for a Common Vulnerabilities and Exposures (CVE) to be determined and finally patched. Thus, to mitigate against cyber-attacks, it is important to understand both known CVEs and unknown vulnerabilities.

In this thesis, we first conducted a large-scale crawling of Git commits for some popular open source repositories like Linux. Second, because there is no prior dataset for security-relevant Git commits, we developed a web-based triage system for security researchers to perform manual labelling of the commits. Finally, after the commits are cleaned and labelled, a deep neural network is implemented to automatically identify vulnerability-fixing commits (VFC) based on the commit messages. The approach has achieved significant better precision than state-of-the-art while improving the recall rate by 16.8%. In the end, we present a thorough quantitative and qualitative analysis of the results and discuss the lessons learned and room for future work.

Acknowledgement

First and foremost, I would like express my sincere gratitude towards my supervisor Professor Liu Yang for his continuous support of my master study. He has been not just a supervisor, but also a mentor, career advisor and a friend. With his constant encouragement and support, I was motivated to produce the best performances within my ability.

I would like to give special thanks to Dr. Wang Chenyu for lending his expertise in software security, Dr. Zhou Yaqin for helping me understand various deep learning techniques and providing precious advice to my research. I would also like to thank Ph.D. student Jing Kai Siow for his assistance on the commits crawler and Krithika Sundararajan for her work on the web-based triage system. Of course the research would not be possible without all the help and support from my colleagues at Scantist.

Last but not least, I would like to thank my parents for their unconditioned love and support. This thesis is dedicated to them.

Wan Liuyang

January 2019

Table of Contents

Abstract ..... *i*
Acknowledgement ..... *ii*
Table of Contents ..... *iii*
List of Figures ..... *v*
List of Tables ..... *vi*
1. Introduction *1*
1.1 Background and Motivation 1
1.2 Contributions of the Thesis 3
1.3 Project Schedule 4
1.4 Thesis Organization 4
2. Literature Review *6*
2.1 Introduction 6
2.2 Static Analysis 6
2.3 Dynamic Analysis 7
2.4 Symbolic Execution 7
2.5 Software Metrics 8
2.6 Machine Learning 9
2.7 Summary 9
3. Data Collection and Pre-processing *11*
3.1 Introduction 11
3.2 Commits Crawling 12
3.3 Keyword-based Filtering 14
4. CVE Triage System and Dataset Building *16*
4.1 Background 16
4.2 Functional Requirement 16
4.3 Web Application Implementation 19
4.4 Deployment 20
4.5 Manual Labelling Process 20
5. Deep Learning for Commit Messages *22*
5.1 Background 22
5.2 Word Embedding Techniques 23
5.2.1 Background 23
5.2.2 Bag of Words 23
5.2.3 TF-IDF 24
5.2.4 word2vec 24

5.2.5 Training word2vec Model 26
5.2.6 Evaluation of Trained word2vec Model 27
5.2.7 Using Pre-trained GloVe Model 27
5.3 Deep Neural Network 28
5.3.1 Background 29
5.3.2 LSTM 29
5.3.3 Proposed Deep Neural Network 29
6. Results Discussion 32
6.1 Word2vec Performance Evaluation 32
6.2 Overall System Performance Evaluation 33
7. Conclusion and Future Work 36
References 37
Appendix A - Keywords for Filtering (by Language) 40
Appendix B – Repository List for Commits Crawling 41

List of Figures

Figure 1 Percentage increase in total packages indexed (October 2016 – October 2017)......... 1

Figure 2 NVD: number of CVEs by year 3

Figure 3 Data collection overall system architecture 11

Figure 4 Commits crawling process 12

Figure 5 Example VFC for Linux project (9824dfa) 13

Figure 6 Use case diagram for web-based CVE Triage System 17

Figure 7 Screenshot: triage main page 18

Figure 8 Screenshot: assignment of triage tasks to different users 18

Figure 9 Screenshot: bulk upload from CSV file 18

Figure 10 Screenshot: triage summary 18

Figure 11 Manual labelling process 21

Figure 12 Word2vec CBOW model 25

Figure 13 Word2vec Skip-Gram model 25

Figure 14 GloVe format of word representation 28

Figure 15 word2vec format of word representation 28

Figure 16 Code snippet for converting GloVe format to word2vec 28

Figure 17 RNN have loops 29

Figure 18 Deep neural network for commit messages 30

Figure 19 Word2vec model performance by Google Test 32

List of Tables

Table 1 Project timeline 4

Table 2 Open source project Linux on GitHub 11

Table 3 A Commit object breakdown 14

Table 4 Keyword filtering results for C projects 15

Table 5 Keyword filtering results for other languages projects 15

Table 6 Database schema of CVE Triage (only partial schema is shown) 19

Table 7 Labelling results for C projects 21

Table 8 Labelling results for other languages projects 21

Table 9 Evaluation results on Linux dataset 33

Table 10 Evaluation results on FFmpeg dataset 33

Table 11 Evaluation results on Qemu dataset 34

Table 12 Evaluation results on WireShark dataset 34

Table 13 Evaluation results on combined C dataset 34

Table 14 Evaluation results on combined other languages dataset 35

Table 15 Keywords for C 40

Table 16 Keywords for Java 40

Table 17 Keywords for JavaScript/Python/Ruby/Go 41

Table 18 Java Repository List (Top 100 results) 41

Table 19 JavaScript/Python/Ruby/Go Repository List (Top 100 results) 41

1. Introduction

1.1 Background and Motivation

Vulnerabilities, in the context of software security, are specific flaws or oversights in a piece of software that allow attackers to achieve malicious goals, such as exposing or altering sensitive information, disrupting or destroying a system, or taking control of a computer system or program [1]. Software vulnerabilities are the root causes of the cyber-attacks today. There are numerous notable instances of security breaches due to known vulnerabilities in software system. One of the most famous examples is the Equifax breach, which has been depicted as the most economically damaging hack in U.S. history [2], where the hackers took advantage of a known vulnerability in the open-source project Apache Structs Web application framework. The hack has resulted in the massive exposure of personal information of 143 million US citizens, including credit cards and social security number, along with a 20 billion loss in the capital market. What stands out is the timeline of the incident. The vulnerability was first publicly disclosed on March 6, 2017. It took only one day for the exploit script to appear in the wild and the breach to finally happen in two months.

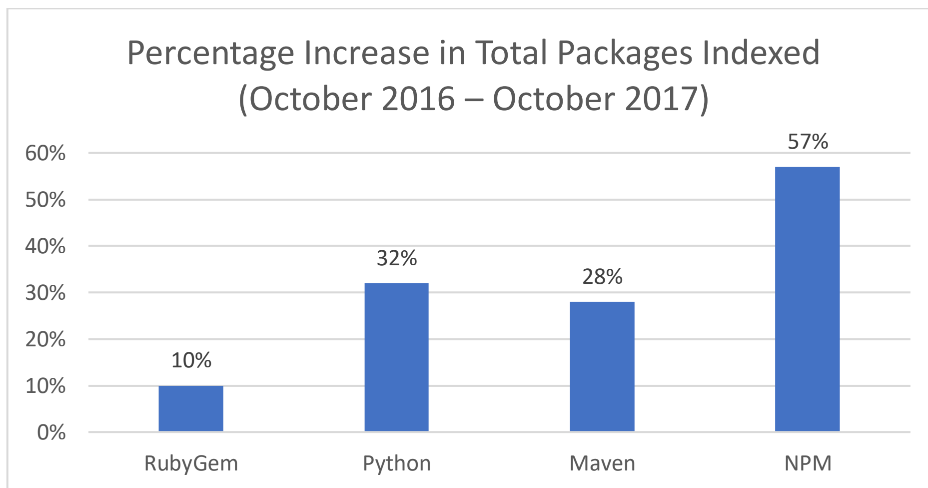


Figure 1 Percentage increase in total packages indexed (October 2016 – October 2017)

The trend of adopting open source software to build software system is ever-increasing. According to a recent survey [3] on the state of open-source security, more than 80% of all commercial software developers use open-source components within their applications. It also has seen an increase in the open source packages index, for example, 10% increase in RubyGem[1] for Ruby, 32% increase in PyPI[2] for Python, 28% increase in Maven[3] for Java and 57% increase in NPM[4] for JavaScript. With the increase of open source libraries, the number of the vulnerabilities is also growing. Concretely, the number of open source application library vulnerabilities increased by 53.8% in 2016 and additional 39.1% in 2017. The U.S. government has created the National Vulnerability Database (NVD) [4] to serve as the single source of truth for vulnerabilities. Started in 1999, NVD has grown to over 81,000 Common Vulnerabilities and Exposures (CVE) and shows no signs of slowing down. In the year of 2017 alone, more than 8,500 CVEs have been reported, which is more than any other year in history. However, the coverage of the vulnerabilities can be an issue, depending on the ecosystem. For example, NVD only tracks 67% of RubyGem vulnerabilities and 11% of NPM vulnerabilities. Another issue is the pace of vulnerability discovery. For example, it frequently takes a long time for a vulnerability to make its way into the NVD database and gets assigned a Common Vulnerability Scoring System (CVSS) score and a Common Platform Enumeration (CPE). Therefore, we need a system to automatically detect vulnerabilities before NVD disclosure so that patches can be made in time to prevent damages.

[1] https://rubygems.org/
[2] https://pypi.org/
[3] https://maven.apache.org
[4] https://www.npmjs.com/

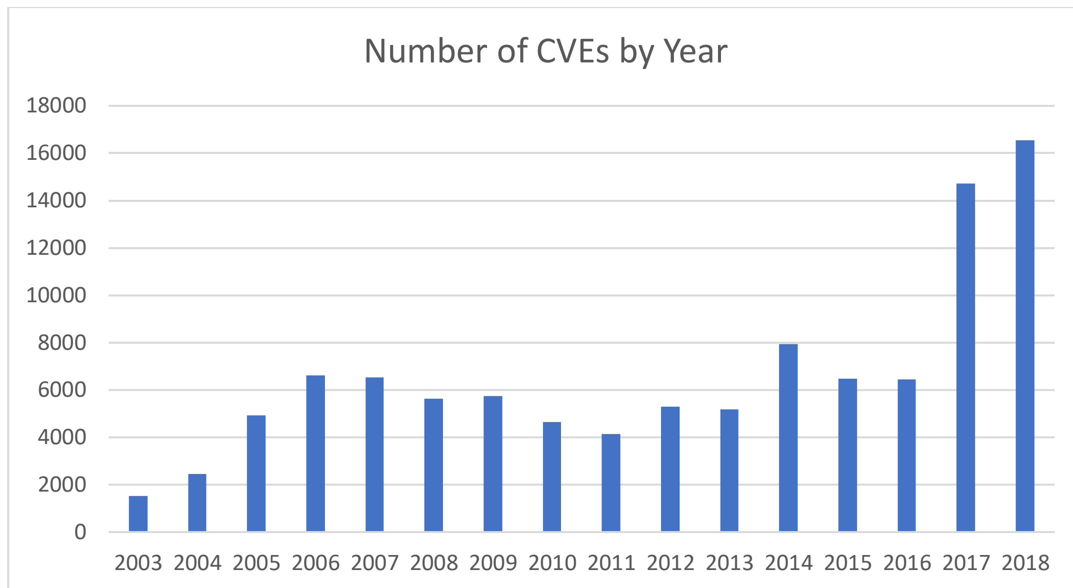


Figure 2 NVD: number of CVEs by year

While vulnerabilities information are tracked by NVD, most of the developer activities took place in Version Control System (CVS) like Git and SVN, out of which GitHub[5] is the most popular platform for developers worldwide. In CVS like Git, each developer contributes to an open source repository via commits, which contain information like commit message, files changed, author of the commit and so on. Therefore, commit is the natural smallest unit to study where vulnerabilities are being introduced and fixed.

In this thesis, we aim to build an automated system to identify undisclosed vulnerabilities via Git commits. There are three main types of commits: vulnerability-contributing commits (VCC), vulnerability-fixing commits (VFC) and vulnerability-unrelated commits. In this thesis, we mainly focus on finding VFC via public available Git Commits on GitHub. Once we can locate the VFC, it is easy to infer the VCC because the VFC usually contain code changes to fix a bug or vulnerability in the VCC.

1.2 Contributions of the Thesis

[5] https://github.com/

There are three major contributions in this thesis: (1) we built the first-ever vulnerability-fixing commits database based on crawling and labelling huge amounts of Git commits from GitHub, not only as the training data for our work but also aim to provide a baseline for future research (2) we designed and implemented a web-based triage system that can help security researchers to label commits and save the amount of manual labour, which does not scale well and tend to be error-prone (3) we developed an automated deep neural network based VFC detection system that outperformed state-of-the-art. The system aims to provide security researchers with an early indication at a high level of confidence that a commit is indeed security-related.

1.3 Project Schedule

The timeline of the project is summarized in Table 1.

S/N	Project Activity	Days	Start Date	End Date
1.	Requirement Analysis and Specification	7	15/01/2018	22/01/2018
2.	Literature survey and review	14	22/01/2018	05/02/2018
3.	Building web-based triage system	30	05/02/2018	05/03/2018
4.	Data collection and manual labelling	240	05/03/2018	11/01/2018
5.	Deep learning model design and implementation	60	01/11/2018	31/12/2018
6.	Parameters tuning for deep learning models and performance evaluation	14	31/12/2018	14/01//2019
7.	System delivery	7	14/01/2019	21/01/2019

Table 1 Project timeline

1.4 Thesis Organization

This thesis is organized into the following chapters:

Chapter 1: we introduce the background and motivation of the research and highlight the major contributions of the thesis.

Chapter 2: we perform the literature review and examine various prior work, along with comparison and summary of each approach.

Chapter 3: we describe the data collection process and methodology of building the commits database, including our novel keyword-based filtering techniques.

Chapter 4: we illustrate the design and implementation details of the web-based triage system, and explain how it is used for manual data labelling. We also touch upon the rationale for choosing various tools and briefly describe the deployment process.

Chapter 5: we discuss various deep learning techniques used to build our automated vulnerability detection system, such as word embedding and use of Recurrent Neural Network.

Chapter 6: we share our experiment outcome, discuss the findings and compare our results against the state-of-the-art for evaluation.

Chapter 7: we draw conclusion of the research and provide some thoughts on possible future work.

2. Literature Review

2.1 Introduction

In this section, we examine a broad range of techniques used in literature for digging vulnerabilities in software system, including static analysis, dynamic analysis, symbolic execution and machine learning. We also compare the pros and cons of each of these approaches and discuss how they are related to our proposed deep learning methods.

2.2 Static Analysis

Static analysis is the method of debugging computer program without actually executing. It typically involves analyzing a set of code against a set (or multiple sets) of coding rules and using techniques such as Taint Analysis and Data Flow Analysis to attempt to highlight potential vulnerabilities. As a result, they can be very useful for detecting specific types of flaws such as buffer overflows and SQL Injection. As static analysis scales well with large amounts of code, it is usually performed at the code review stage or as an Integrated Development Environment (IDE) plugin before the code is even committed to Version Control System (VCS). The majority of the static analysis tools have been focused on analyzing software written in high-level languages, e.g. FlawFinder [5], Splint [6], Microsoft PREfast [7] for C/C++, FindBugs [8] for Java and PMD [9] for multiple languages (Java, JavaScript, XML). Commercials tools like Fortify [10] (Java, C#, C, C++, Swift, PHP) and Coverity [11] (Android, C#, Node.js, Objective-C, Scala, Swift) are also available for much more complicated configurations and thus longer running time. However, these tools perform only shallow understanding of the code and may produce many false positives. For example, the author of FlawFinder [5] states that the tool primarily uses simple text pattern matching and does not understand the semantics of the code at all and furthermore advises to use the tool only as guide in finding and removing security vulnerabilities.

2.3 Dynamic Analysis

In contrast to static code analysis, dynamic analysis analyzes the program by executing it with actual inputs. Techniques like fuzzing works by feeding random data as input to a software system in the hope to crash the system and expose a vulnerability. For example, Holler et al. [12] used fuzzing on code fragments to detect vulnerabilities in Mozilla JavaScript and PHP interpreters. Papagiannis et al. [13] present PHP Aspis, a source code transformation tool that applies partial taint tracking at the language level, to track Cross Site
Scripting and SQL Injection vulnerabilities. It carries out the taint propagation in the third-party plugins to achieve the highest performance and had successfully detected many injection exploits that were found in PHP-based WordPress plugins. Cho et al. [14] present MACE, a tool for state-space exploration. It used a combination of symbolic and
concrete execution to build and refine an abstract model for the analyzed application. The results show a good code coverage and exploration path, and the tool was able to find a few vulnerabilities in open-source projects. Dynamic analysis can be useful in discovering unknown vulnerabilities, however the long running time and hassle of setting up the full working environment is not cost-efficient compared to our approach in this thesis.

2.4 Symbolic Execution

Symbolic execution is a software technique that is useful to aid the generation of test data and assure the program quality by exercising various code paths in a target system.
Unlike dynamic analysis, which runs the target system with pre-defined values, symbolic analysis runs the system with input values replaced by symbolic variables whose values can be anything. Cadar et al. [15] present KLEE, an open-source symbolic execution tool capable of automatically generating tests with high coverage on a complex and environmentally-intensive programs. When also used as a bug finding tool, it was able to detect 56 serious bugs in

scanning 452 applications, out of which some bugs are left un-discovered for over 15 years. However, the limitation of KLEE is that it requires manual annotation and modification of the code, which itself requires strong domain knowledge. Furthermore, like most symbolic execution tool, the runtime complexity grows fast with the increase of path numbers, which eventually will lead to path explosion.

2.5 Software Metrics

Many previous works [16] [17] [18] [19] [20] [21] [22] look at various software metrics to mine vulnerabilities from software repositories, e.g. code-churn, code-complexity, coverage, developer-activity, GitHub metadata. For example, Perl et al. [17] performed the study of using metadata in code repositories along with code metrics to find vulnerability-contributing commits (VCC) in open source projects. The authors assert that most open-source projects are managed by Version Control System (VCS) such as Git, Mercurial, CVS or Subversion, and therefore commit is the natural units to check for vulnerabilities. They've conducted the first large-scale mapping of CVEs to VCC and build a dataset containing 66 C/C++ projects and a total of 170,860 commits, including 640 VCC mapped to relevant CVE IDs. The dataset is also published by the authors for free to the public. Based on the dataset, they've trained a Support Vector Machine (SVM) classifier to flag suspicious commis which outperformed FlawFinder [5] by reducing false positives by 99% while maintaining the same recall rate. This is very close to our work, but looking at the problem at a slightly different angel, as our work mainly focus on finding VFC instead of VCC. The use of SVM also means this method does not generalize well for languages other than C/C++, as the feature extraction and training will need to be re-done for the other languages. Meneely et al. [23] studied the open-source project Apache HTTP web server by exploring the VCC and found over 100 VCCs based on metrics like developer activities and code-churn. Their study presents a few interesting findings: (1) code-churn activities are empirically associated with VCCs, such that bigger commits are more

likely to introduce vulnerabilities (2) commits that are affected by more developers are more likely to be vulnerabilities (3) commits authored by new contributors are more likely to be VCCs.

2.6 Machine Learning

In recent research, machine learning and data mining techniques have been widely in the identification of vulnerabilities. Many previous works use a combination of software engineering metrics and traditional machine learning algorithms to build a vulnerability predictor [19] [24] [25]. These methods perform the analysis on the source code and often rely heavily on the domain expertise for feature engineering. To remove the need for analysis designers, Chae et al. [26] proposed a technique to automatically generate features for learning program analysis heuristics for C-like languages. The method uses a powerful program reducer and an efficient method for building data-flow graphs to reduce and abstract programs. The reduced programs are then matched some queried to get feature vectors. However, this approach does not apply to other programming languages and may fail for some classification tasks with long-term dependencies. Ponta et al. [5] built a dataset of VFC for open-source projects which maps 625 publicly disclosed vulnerabilities onto 1282 VFC, covering 205 distinct projects. The data was obtained from a combination of NVD and project-specific websites. They further used to the data to train classifiers that could automatically determine security-relevant commits in the source code repository.

2.7 Summary

In this section, we have reviewed and compared a variety of different techniques in the field of vulnerability detection.While most of the reviewed work focus on finding new vulnerabilities, our work is related to detection of existing vulnerabilities that remain unidentified. Thus, the work most similar to ours are [17] and [27], which classify a commit as vulnerable or not based

on the commit messages (and some other metrics). We will compare our results with these two approaches in the evaluation section. Both our approach and [5] performed extensive data gathering to hand-curate the dataset. However our dataset covers much more VFC and wider range of programming languages whereas [5] only considered Java. The dataset was also heavily focused on industry relevant projects which is not a good indicator of the whole open-source scene in the real world.

Another common challenge faced by most machine learning based approaches is the imbalance class data problem. Because by nature vulnerabilities are few and sparse in a dataset, which can greatly hinder the performance of the machine learning algorithms. For example, Zhou and Sherma [27] found that vulnerabilities-related commits are less than 10% of the entire commits; to address the highly imbalance nature of the data, they used a probability-based K-fold stacking algorithm which ensembles multiple individual classifiers, like random forest (RF), Gaussian Naive Bayes (GNB), K-nearest neighbors (KNN), linear SVM, gradient boosting (GB), and AdaBoost (Ada).

3. Data Collection and Pre-processing

3.1 Introduction

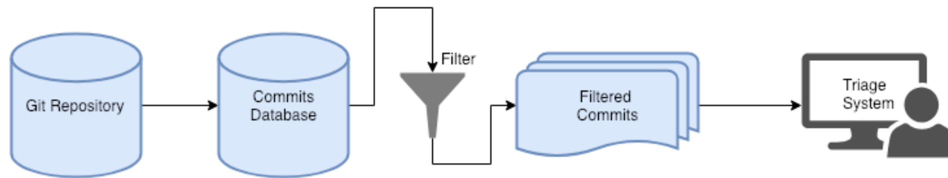


Figure 3 Data collection overall system architecture

In our research, we are interested in the extracting information about unknown vulnerabilities from GitHub. First of all, a team of security researchers carefully select a list of target open-source repositories, based on a few factors such as the popularity of the software and the relevance of the software in security perspective. We developed a crawler to download all the commits into a local database. For smaller projects, we simply crawl all the commits within those projects; For larger collaborative projects like Linux, where the total number of commits is huge (more than 810, 000 commits for Linux), we only extract commits within a specific time frame to balance the dataset.

torvalds / linux — Watch 6,598 — Unstar 67,864 — Fork 24,525

Code — Pull requests 247 — Releases 587 — More

Linux kernel source tree

810,299 commits — 1 branch — 587 releases — ∞ contributors — View license — 2.25 GB

Branch: master — Create new file — Find file — Clone or download

sean-jc and torvalds mm/mmu_notifier: mm/rmap.c: Fix a mmu_notifier range bug in try_to_un... — Latest commit ba42273 11 hours ago

Documentation	Merge tag 'csky-for-linus-5.0-rc1' of git://github.com/c-sky/csky-linux	18 hours ago
LICENSES	Merge tag 'docs-4.20' of git://git.lwn.net/linux	3 months ago
arch	Merge tag 'csky-for-linus-5.0-rc1' of git://github.com/c-sky/csky-linux	18 hours ago

Table 2 Open source project Linux on GitHub

As the number of commits are huge and not all commits are necessary security-related. We have devised an innovative keyword-based mechanism to filter out the commits that are irrelevant to our study.

After the filtering process, a reasonable amount (10%-30%) of the raw commits are left. Before these commits are ready for hand labelling, we manually examined and cleaned the data by removing irrelevant tokens in the commit message. Once the commits are filtered and cleaned, they are imported into the web portal for manual labelling. To ensure the quality of the human labelling, each commit will go through three rounds of rigorous triage before it is determined to be a VFC or non-VFC. Finally these manually labelled commits will be used as the dataset to train our deep learning model.

3.2 Commits Crawling

Figure 4 Commits crawling process

First of all, we have hand-curated a list of open source repositories to crawl. The list is stored in the database and regularly updated by our security experts. We have selected a wide range of open source projects across multiple programming languages: C, C++, Java, Python, Ruby, Go. For C language, we chose four popular and diversified open source libraries: FFmpeg, Linux, Qemu and Wireshark. Except for Linux, all the history commits of the project are crawled. Due to the large amount of commits history of Linux, we only crawled commits between 2016 and 2017 to balance the dataset.

There are two approaches for downloading commits from GitHub: *online* and *offline*. For the online approach, we use a RESTful API tool like requests[6] to extract the commits information. The advantage of this approach is that no additional disk space is required to store all the commit files. However, this approach will suffer if the number of commits is huge and can

[6] http://python-requests.org

only handle a small amount of commits at a time. Therefore we turn to the offline approach: we perform a deep (as opposed to shallow) clone of all the Git repositories.

Shallow Cloning. A shallow clone is a repository created by limiting the depth of the history that is cloned from an original repository. The depth of the cloned repository, which is selected when the cloning operation is performed, is defined as the number of total commits that the linear history of the repository will contain. A shallow clone is created using the --*depth* option when calling the clone command, followed by the number of commits that you want to retrieve from the remote repository. When the --*depth* option is omitted, by default it will download all the commits in history.

```
$ git clone --depth=1
```

The standard way is to use the Git command line tool. In our work, we use a Python wrapper for Git command called GitPython[7] to perform the commits downloading. A Git Commit object contains rich information about the actual code changes as well as an abundance of metadata information. Figure 5 shows an example of a Git commit (VFC for CVE-2018-20511) on GitHub.

Figure 5 Example VFC for Linux project (9824dfa)

[7] https://github.com/gitpython-developers/GitPython

The Git Object can be further broken down in Table 3 .

Type	**Example**	**Remarks**
message	*net/appletalk: fix minor pointer leak to userspace in SIOCFINDIPDDPRT\n\nFields ->dev and ->next of struct ipddp_route may be copied to\nuserspace on the SIOCFINDIPDDPRT ioctl. This is only accessible\nto CAP_NET_ADMIN though. Let's manually copy the relevant fields\ninstead of using memcpy().\n\nBugLink: http://blog.infosectcbr.com.au/2018/09/linux-kernel-infoleaks.html\nCc: Jann Horn <jannh@google.com>\nSigned-off-by: Willy Tarreau <w@1wt.eu>\nSigned-off-by: David S. Miller <davem@davemloft.net>*	Summary of the commit. The first line usually serves as the subject of the commit message. It can also contain additional information like code change reviewer and approver.
sha	*9824dfae5741275473a23a7ed5756c7b6efacc9d*	A 40-character checksum hash. It is the SHA-1 checksum of the content
files changed	*[drivers/net/appletalk/ipddp.c]*	A list of files being changed
parent	*018349d70f28a78d5343b3660cb66e1667005f8a*	A pointer to the sha of the parent commit
branch	*master*	A Git branch on which the commit is made
tag	*v5.0-rc1*	A Git tag associated with the commit, typically associated with a release
author	*Willy Tarreau*	Author of the code changes, may not have access to the project. For open source project this can be anyone.
committer	*davem330*	Whoever has write access to the repository, typically the collaborator of the project on an open source project.

Table 3 A Commit object breakdown

3.3 Keyword-based Filtering

Each chosen project contains over 50,000 raw commits. The amount is still massive for human labelling and only less than 30% percent are related to vulnerability fixing. Therefore we need a way to extract only security-relevant commits from the raw commits. A keyword-based filtering mechanism is used. We first collected a list of security-related keywords from NVD database and filter out the commits that do not relate to any of the keywords.

After the filtering process, only around 8.2% of the commits are left for Linux and only 16% to 21% commits are left for other C projects. For other languages, less than 12% of the commits remained for Python and Go, and around 20%-30% are left for Java, JavaScript and Ruby. In total, we have collected **50,000** commits for C/C++, **10,000** commits for Java and **5,000** commits for other languages combined (JavaScript/Python/Ruby/Go).

Project	Total	Matched	Matched Ratio
Linux	165714	13290	0.082
FFmpeg	82361	13672	0.166
Qemu	57700	12348	0.214
WireShark	62987	10141	0.161

Table 4 Keyword filtering results for C projects

Language	Total	Matched	Matched Ratio
Java	48717	9987	0.205
Python	11951	1482	0.124
JavaScript	5669	1661	0.293
Ruby	4627	1203	0.260
Go	5581	653	0.117

Table 5 Keyword filtering results for other languages projects

The commits are then imported to the web-based CVE Triage system for manual labelling. The implementation details of the CVE Triage will be discussed in Chapter 4. The full keywords list used for filtering for C, Java and other languages can be found at **Appendix A**.

4. CVE Triage System and Dataset Building

4.1 Background

Building a high quality dataset is the key to the success of any machine learning problem. In this thesis, our first challenge is the lack of VFC database in prior work. So we have set up a full pipeline as described in Chapter 3 to collect massive amounts of commits from GitHub. Due to time and manpower constraint, we are unable to build large-scale datasets like ImageNet[8] , so our goal is to build a medium-sized dataset of vulnerability-fixing commits for our work and also as the baseline for future research.

To fully make use of computers to assist the hand-labelling process, we have designed the triage system to be a web application, so that all users with permission to the web application can perform the triage tasks without geographical limitation. The triage result will go into a shared database to avoid any duplicate assignments.

As the security triage process requires strong domain expertise and accuracy is of great importance, we've implemented a three-stage triage system to make sure the final status is as accurate as possible.

4.2 Functional Requirement

The full functional requirements of the web-based triage system are as follow:

- The web application shall allow users to sign up with a GitHub account
- The web application shall allow users to log in with a GitHub account
- The web application shall load commits information from a database and display the contents in a table with pagination

[8] http://www.image-net.org/

- The web application shall allow users to filter commits by repository name, date, commit message and triage status
- The web application shall have an interface for users to perform bulk upload of commits via a Comma-separated values (CSV) file
- The web application shall be able to display the real-time summary of all triage results
- The web application shall allow site admins to manually assign commits to other users in the system for triage
- The web application shall have a basic permission system to allow only privileged users (senior security experts) to perform final confirmation

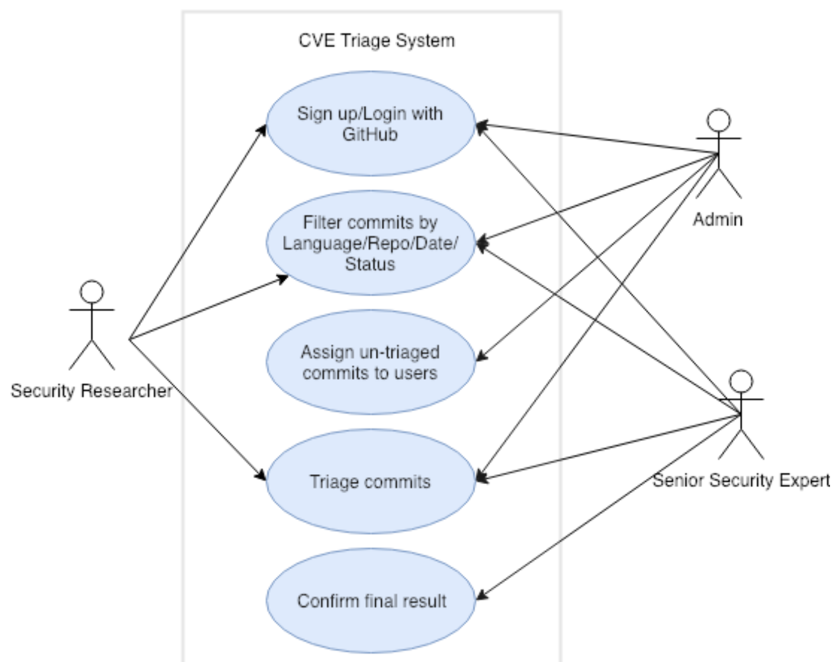


Figure 6 Use case diagram for web-based CVE Triage System

Security Bugs Triage Commits Assignment Triage Summary Deep Learning Ruleset sfdye | Sign Out

Triage Home

Status: --------- Repo: --------- Assignee: --------- Date From: Date To: Ruleset: Exclude Rules:
Commit Message: Ordering: --------- Submit

Unsure: 11530 Not A Vulnerability: 37385 Vulnerability: 58448 Incomplete: 26 Unique Incomplete: 25 Unassigned Commits: 52957

Previous 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 Next

Repo	CommitId	Message	Files Changed	Ruleset	Assigned	Edit Result	Triage Result
gst-plugins-good	b310393916467e0a24d04c1372132773c371a2ed git://anongit.freedesktop.org/gstreamer/gst-plugins-good	**opuspay: fix timestamps** opuspay: fix timestamps Copy timestamps to payloaded buffer. Avoid input buffer memory leak. Fixes https://bugzilla.gnome.org/show_bug.cgi?id=692929	['gst/rtp/gstrtpopuspay.c']	https://bugzilla, leak	**yaqinZhou** April 25, 2018, 11:31 p.m.	Triage Assessment: Incomplete / Vulnerability / Not A Vulnerability / Unsure; Comment; Submit	Triages Completed: **2 / 2**; Vulnerability May 9, 2018, 6:49 p.m.

Figure 7 Screenshot: triage main page

Security Bugs Triage Commits Assignment Triage Summary Deep Learning Ruleset sfdye | Sign Out

Assignment Summary

Date From: Date To: Submit

** Unique counts are indicated in brackets ()

User	Total Assigned (All Time)	Total Incomplete (All Time)	Total Completed	Vulnerability	Not A Vulnerability	Unsure
chan0415	7	7 (7)	0	0	0	0
dangokyo	16640	0 (0)	16640	9547	6227	866
DCHIA017	1836	0 (0)	1836	1328	357	151
HarDToBelieve	17045	0 (0)	17045	4625	3889	8531
hongphipham95	796	0 (0)	796	214	582	0
krithika369	1	0 (0)	1	1	0	0
ksp463	110	0 (0)	110	57	53	0

Figure 8 Screenshot: assignment of triage tasks to different users

Upload Projects Projects Bulk Upload

Bulk Upload

Upload multiple repos/projects through **csv files** only

The **csv files** you upload should contain the columns below:

name: String, required | **repo_url**: String, required | **homepage_url**: String, Nullable | **branch**: String, Nullable | **platform**: String, Nullable | **scm_type**: (git, svn, cvs, hg) | **language_id**: Integer, (0 - 'Unknown', 1 - 'C/C++', 2 - 'Java')

Choose file No file chosen

Figure 9 Screenshot: bulk upload from CSV file

Security Bugs Triage Commits Assignment Triage Summary Deep Learning Ruleset

Triage Result Summary

Assessed Commit Labels	Count	% of All Assessed Commits
Consistent / Vulnerable	37203	37.38
Consistent / Not Vulnerable	16205	16.28
Inconsistent	26352	26.47
Unsure / Vulnerable	8023	8.06
Unsure / Not Vulnerable	11056	11.11
Unsure / Unsure	698	0.7

Figure 10 Screenshot: triage summary

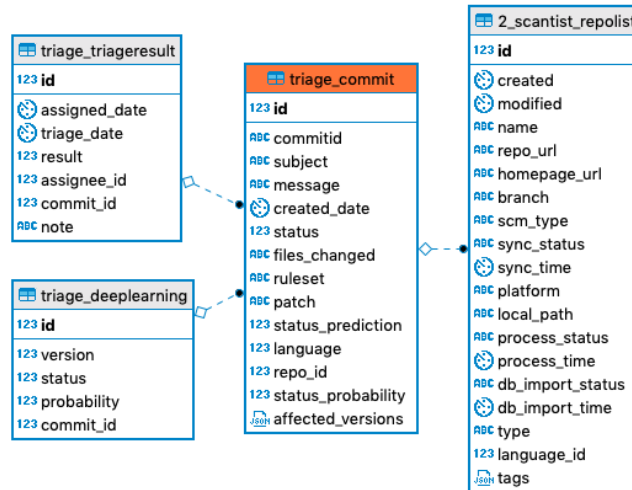


Table 6 Database schema of CVE Triage (only partial schema is shown)

4.3 Web Application Implementation

We use Python as the programming language of choice for other parts of the research: GitPython for the commits crawler, TensorFlow for deep learning training. In order to have a consistent codebase, we pick Django[9] as the framework for our web-based triage system. It is a high-level Python web framework for rapid development with a clean and pragmatic design. Specially, Django follows the Don't Repeat Yourself (DRY) principal to allow us reuse many common tasks and focus on the application logic itself. The framework also ships with loads of built-in protection against some of the most common web vulnerabilities such as SQL Injection, Cross-Site Scripting and Cross-Site Request Forgery.

The overall architecture can be seen as a classic MVC (model/view/controller), while Django has its own nomenclature and define the callable objects generating the HTTP responses "views". In the "models", we define the type of classes along with their attributes we want to store in the database, which then will be connected by the built-in Object Relational Mapper (ORM) for us to make DB queries. In the "controllers", we program the core logic, such as the

[9] https://www.djangoproject.com/

user authentication, database queries, caching and store the values in the HTTP context dictionary. Finally, in the "views", we load the values from the HTTP context and render them on the HTML page, together with JavaScript for animation and CSS for styling.

To start off our project, we use cookiecutter-django[10] to generate the boilerplate code. The cookicutter-django is an engine that automatically generates production-ready Django projects with many useful features and best practices, such as 12-Factor[11] compliant, Twitter Bootstrap support, automatic SSL/TLS certificates retrieval from Let's Encrypt[12], Docker support and AWS integration. For details of Django best practices please refer to [28].

4.4 Deployment

We use Docker to encapsulate all the runtime dependencies and ensure the parity between the development and production environments. For production, we deployed our dockerized containers to a single Amazon Web Services (AWS) *r4.2xlarge* EC2 instance. For database, we used one AWS Managed relational database service (RDS) *db.t2.large* instance of PostgreSQL deployed in the *ap-southeast-1* region. The web application is served by Nginx frontend which will proxy the dynamic contents to be served by the application server Gunicorn[13] and static contents to be served by Nginx itself.

4.5 Manual Labelling Process

[10] https://github.com/pydanny/cookiecutter-django
[11] https://12factor.net/
[12] https://letsencrypt.org/
[13] https://gunicorn.org/

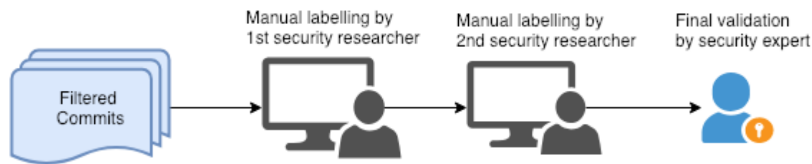


Figure 11 Manual labelling process

When commits are imported into the triage system, they are ready to be distributed to different security researchers. To make sure the data is of high quality, we enforced a three-stage triage workflow as follow:

1. First security research to examine the commit, label the commit as Vulnerability/Not a Vulnerability/Needs Further Investigation and provide additional comments if any.
2. Second security research to examine the commit, label the commit as Vulnerability/Not a Vulnerability/Needs Further Investigation and provide additional comments if any.
3. Senior security expert to confirm the final status of the commit; if there is a discrepancy, the status will be brought to the team for further discussion.

Project	VFC	Non-VFC	VFC Ratio
Linux	8762	4528	0.659
FFmpeg	6002	7670	0.439
Qemu	4853	7495	0.393
WireShark	3803	6338	0.375

Table 7 Labelling results for C projects

Language	VFC	Non-VFC	VFC Ratio
Java	5832	4155	0.584
Python	522	960	0.352
JavaScript	732	929	0.441
Ruby	416	787	0.346
Go	179	474	0.274

Table 8 Labelling results for other languages projects

5. Deep Learning for Commit Messages

5.1 Background

We can divide commits into three categories, i.e. vulnerability-contributing commits (VCC), vulnerability-fixing commits (VFC) and vulnerability-unrelated commits where (1) a vulnerability-contributing commit is the commit that introduces the vulnerability in the code base (2) a vulnerability-fixing commit is the commit that removes or fixes the vulnerability which has been introduced in the corresponding vulnerability-introducing commit (3) a vulnerability-unrelated commit neither introduces nor fixes a vulnerability. When a commit is both a vulnerability-introducing commit and a vulnerability-fixing commit, we will treat it as a vulnerability-fixing commit.

In this thesis, we are given a commit message and need to determine whether it is a vulnerability-fixing commit or not. This is a classification problem and in machine learning, it can be formulated as a function:

$$F(x_i, \theta) = p_i \tag{1}$$

In the above equation, x_i is the input commit and p_i is the probability that x_i fixes a vulnerability, and θ is the model parameter that the deep learning algorithm targets to learn. However, before the commit x_i is fed into any machine learning algorithm, it has to be converted into a vector representation using some word embedding techniques. The word embedding techniques will be discussed in the following section.

Since our problem is a binary (i.e. classifying a commit as vulnerability or not a vulnerability) classification problem, it is common to use the cross-entropy loss function as the objective function [32] :

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \tag{2}$$

Here n is the number of training examples and y_i is the label of the commit x_i.

5.2 Word Embedding Techniques

5.2.1 Background

The raw commit message contains only natural language produced by human. In order for our machine learning to understand the text, word embedding is often used as a pre-processing step. Word embedding is the feature learning techniques in Natural Language Processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. In concept, it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension. Here we evaluate multiple word embedding techniques commonly used in literature such as bag-of-words, TF-IDF and word2vec.

5.2.2 Bag of Words

The bag-of-words (BoW) model is an algorithm to count how many times a word appear in a document, disregarding grammar and word order but keeping the frequency. These word counts allow us to compare documents and measure their similarities for applications like search, document classification and topic modelling.

For example, consider the following two sentences:

(1) John likes to watch movies. Mary likes movies too.

(2) John also likes to watch football games.

The bag-of-words of the two sentences can be represented as:

BoW1 = *{"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1}*

BoW2 = *{"John":1,"also":1,"likes":1,"to":1,"watch":1,"football":1,"games":1}*

Sabetta and Bezzi [33] used bag-of-words as the word embedding technique before they use a SVM classifier to study commits that are security-relevant and achieved a precision of 80% and recall of 43%.

5.2.3 TF-IDF

Term frequency–inverse document frequency (or TF-IDF) is another technique to reflect how important a word is to a document in a collection or corpus. The TF-IDF value increases proportionally with the number of times a word appears in the document and is counter-balanced by the number of documents in the corpus that contains the word. This will help to adjust for the fact that some words (like "and", "the") appear more frequently than other words in general. The TF-IDF value of each word is normalized and the total values sum up to one.

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i}) \tag{3}$$

Here $tf_{i,j}$ is the occurrences of i in j , df_i is the number of documents containing i and N is the total number of documents.

5.2.4 word2vec

The bag-of-words and TF-IDF methods work well in some cases, however both of them fail to capture the semantics of words in the context of the corpus, which could be very useful in determining the vulnerability status from the commit message. Word2vec is a popular neural network-based word embedding technique proposed by Mikolov et al. [34], capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

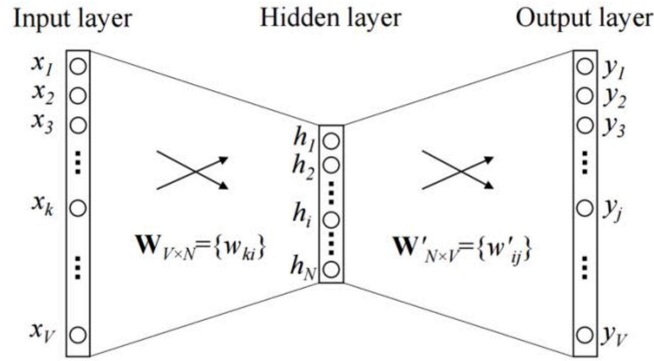


Figure 12 Word2vec CBOW model

It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. From an algorithmic perspective, these two models are similar, except that CBOW predicts target words from source context words, while the Skip-Gram does the inverse and predicts source context-words from the target words. Please refer to [34] for details about word2vec.

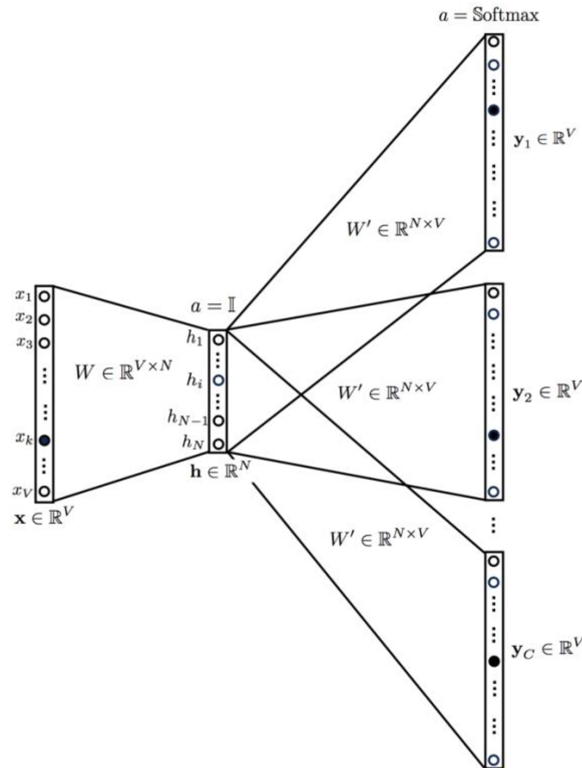


Figure 13 Word2vec Skip-Gram model

5.2.5 Training word2vec Model

For implementation of word2vec, we choose the open-source gensim [35] toolkit, because it utilizes NumPy, SciPy and Cython to achieve high performance computation. Thera are a number of hyperparameters that need to be tuned for the model, including minimum word count, size, downsampling rate, context window size and etc. Here we mainly focus on the following two parameters, which could result in a significant change in the performance of the model. Note that we will use the default CBOW implementation from the gensim toolkit for word2vec (setting the *sg=1* option will switch to Skip-Gram implementation), because for small dataset CBOW tends to produce better results and runs faster.

Min_count. In a big corpus that contains millions or billions of words, word that appears only once or twice are probably typos and garbage [36] and should be eliminated. For example, in some big open source projects, each commit message contains the committer's and approver's name and email. Such information is purely noise and should be removed. The *min_count* is the minimum frequency below which the word should be removed from the internal dictionary. The author of gensim suggests a reasonable value between 0-100, depending on the size of the corpus. If we set *min_count* to a small number, it will ignore irrelevant information and lead to better result; However, if we set the value to high and accidently filter out some rare-seen but useful words (like those library or vulnerability specific keywords), the model won't be able to capture these semantics and the result will be worse. We will be discussing in Chapter 6 in details.

Dimension. It is the size of the neural network layers, or the number of features in the word2vec model. Bigger size requires more training data and longer training time, and could generally lead to better result. In practice, reasonable values are in tens of hundreds. However, if the

training data is not big enough, training with more features will lead to poorer result. We will see this in the results and discussion section.

5.2.6 Evaluation of Trained word2vec Model

As word2vec training is an unsupervised task, there is no good way to objectively evaluate the result. However, Google have released their testing set[14] of about 20,000 syntactic and semantic test examples. Gensim also comes with a script to evaluate the model based on this test set. This number is only a preliminary benchmark for our trained word2vec model. We understand good performance on this test set does not mean word2vec will work well in the application, so the actual performance of the word2vec is evaluated together with our deep neural network.

```
model = gensim.models.Word2Vec.load('path-to-model')
model.accuracy('questions-words.txt')
2019-01-19 22:04:46,711 : INFO : capital-common-countries: 68.2% (90/132)
2019-01-19 22:04:50,196 : INFO : capital-world: 66.7% (108/162)
2019-01-19 22:04:51,443 : INFO : currency: 1.5% (1/68)
2019-01-19 22:05:00,067 : INFO : city-in-state: 38.3% (192/501)
2019-01-19 22:05:04,601 : INFO : family: 83.3% (200/240)
2019-01-19 22:05:09,557 : INFO : gram1-adjective-to-adverb: 17.6% (48/272)
2019-01-19 22:05:09,940 : INFO : gram2-opposite: 20.0% (4/20)
2019-01-19 22:05:23,654 : INFO : gram3-comparative: 76.9% (581/756)
2019-01-19 22:05:30,731 : INFO : gram4-superlative: 72.4% (275/380)
2019-01-19 22:05:35,591 : INFO : gram5-present-participle: 87.6% (333/380)
2019-01-19 22:05:45,112 : INFO : gram6-nationality-adjective: 87.2% (429/492)
2019-01-19 22:06:00,283 : INFO : gram7-past-tense: 60.6% (492/812)
2019-01-19 22:06:17,238 : INFO : gram8-plural: 82.5% (767/930)
2019-01-19 22:06:23,575 : INFO : gram9-plural-verbs: 78.1% (267/342)
2019-01-19 22:06:23,579 : INFO : total: 69.0% (3787/5487)
```

5.2.7 Using Pre-trained GloVe Model

We also discovered GloVe [37], a pre-trained global vectors representation for words by Pennington et al. developed at Stanford University. Figure 14 and Figure 15 show the comparison of word representation when using GloVe and word2vec.

[14] Available: https://raw.githubusercontent.com/RaRe-Technologies/gensim/develop/gensim/test/test_data/questions-words.txt

```
word1 0.123 0.134 0.532 0.152
word2 0.934 0.412 0.532 0.159
word3 0.334 0.241 0.324 0.188
...
word9 0.334 0.241 0.324 0.188
```

Figure 14 GloVe format of word representation

```
9 4
word1 0.123 0.134 0.532 0.152
word2 0.934 0.412 0.532 0.159
word3 0.334 0.241 0.324 0.188
...
word9 0.334 0.241 0.324 0.188
```

Figure 15 word2vec format of word representation

The pre-trained word vectors are open-sourced under the Public Domain Dedication and License:

- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors)
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors)
- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vector)
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors)

Implementation-wise, the genism toolkit has provided a script [38] to automatically convert GloVe vectors to a word2vec compatible model.

```
from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors

glove_file = datapath('glove.txt')
tmp_file = get_tmpfile("word2vec.txt")

from gensim.scripts.glove2word2vec import glove2word2vec
glove2word2vec(glove_file, tmp_file)
model = KeyedVectors.load_word2vec_format(tmp_file)
```

Figure 16 Code snippet for converting GloVe format to word2vec

In our experiment, we use the Common Crawl (*glove.CC.6B.200.txt*) and Twitter (*glove. glove.twitter.27B.200)* pre-trained model to compare with our own-trained word2vec embedding model.

5.3 Deep Neural Network

5.3.1 Background

Among various neural network models, LSTM networks and Convolutional Neural Networks (CNNs) have demonstrated impressive advances in numerous NLP tasks. We will use the two networks to build our learning models. LSTM networks is a particular category of Recurrent Neural Networks (RNNs), competent in learning long-term dependencies of sequences. CNNs have the ability to automatically extract higher-level features to create an informative latent semantic representation of the input for downstream tasks.

5.3.2 LSTM

A common problem with count-based algorithms like BoW and TF-IDF is that they are unable to capture the context of the words. The Long short-term memory (LSTM) is a special kind of RNN that overcomes this problem. In Figure 17, A looks at some input x_t and outputs a value h_t. The RNN can be viewed as multiple copies of the same network, passing information from one network to another.

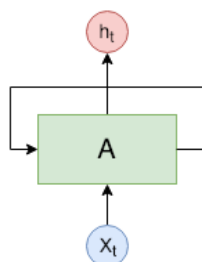


Figure 17 RNN have loops

The chain-like structure make it natural for RNN for handle sequence data, making it perfect for a variety of problems such as speech recognition, language modelling, translation, image captioning, etc.

5.3.3 Proposed Deep Neural Network

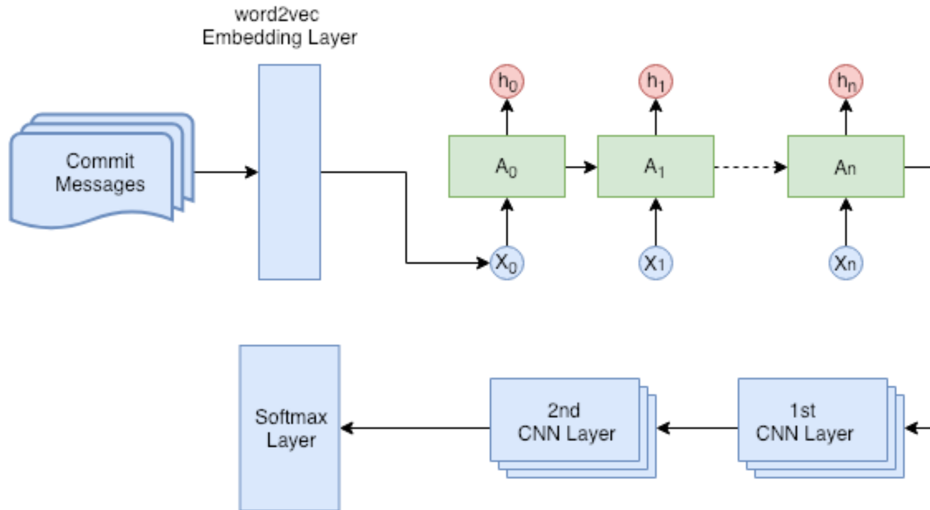


Figure 18 Deep neural network for commit messages

We propose the following commit-message neural network, as depicted in Figure 18.

1) First, a word embedding layer to represent tokenized words of input messages with numerical vectors. Before feeding the raw texts to LSTM layers for feature learning, we firstly encode commit messages in the input layer. For a better representation of commit messages, we pretrain a word2vec model with all commit messages that are not filtered by the keywords. According to Zhou and Sherma [27], using pretrained word2vec models over massive datasets is proven to be more effective in text classification. We examined the performance of word2vec models trained with all the raw commits without filtering and partial commits after the keyword matching, and found that the word2vec model over full commit messages outperforms in all cases.

2) Second, an LSTM layer to extract middle-level representations of commit messages. After the embedding layer, the encoded commit message is then inputted to a LSTM layer to attain middle-level semantic features, where LSTM is designed to capture the long-term dependency of the words. The output of LSTM is then passed into convolutional layers for higher-level features.

3) Third, CNN layers to attain higher-level representations. The convolutional layers aim at extracting abstract representations from the output of LSTM. Compared with the predictors that immediately pass the learned features of LSTM to a softmax layer for classification, we noticed that the predictors with CNN layers performed slightly better.

4) Finally, a softmax layer to output the predicted probabilities. The learned features by the previous CNN layers are directly passed into a softmax layer to determine the likelihood that a commit message indicates the commit fixed a vulnerability or not.

6. Results Discussion

6.1 Word2vec Performance Evaluation

As discussed in section 5.2.6, we conducted a performance evaluation of the trained word2vec model using the Google Test set. Although not decisive, it gives us an idea of how word2vec model will be performing when feeding into the LSTM.

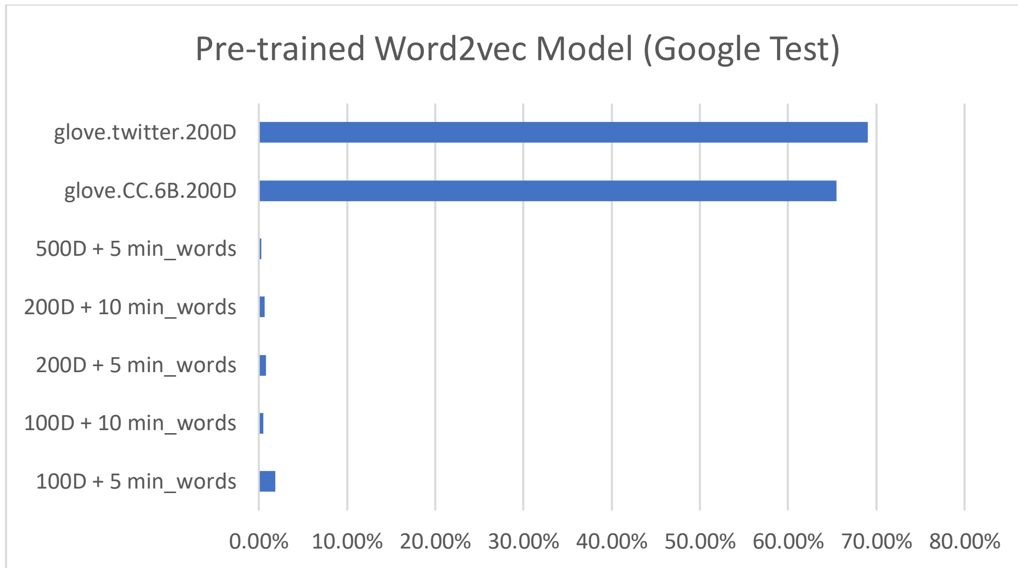


Figure 19 Word2vec model performance by Google Test

As shown in Figure 19, the GloVe pre-trained global vectors significantly outperform our own-trained word2vec in the Google Test. This is expected because the Google Test does not consider the application context and only gauges the accuracy in a global vocabulary. Since the two GloVe models we used in the experiment are trained with massive amounts of data from Wikipedia and Twitter, they naturally contain more words and thus have better results in the Google Test. For our own-trained word2vec models, we can see that the accuracy increases proportionally with the word2vec dimensions; however, when the dimension is too large, the result will be worse again; for the *min_words* parameter, it can be observed that when the number is too large (e.g. 10 compared to the default value 5), the accuracy actually decreases, it shows that some infrequently appeared but useful words are filtered.

6.2 Overall System Performance Evaluation

C/C++ Dataset. We compare our neural network based automated vulnerability detection system **(AVDS)**, with the K-fold stacking (KFS) algorithm [27] and our LSTM network (AVDS without CNN). For the K-fold stacking algorithm, we performed experiments over a full word2vec model (all unfiltered commit messages) and a partial word2vec model (only matched commit messages), to verify the advantage of the full word2vec model over the partial word2vec model. Overall, Table 9-13 show that AVDS beats the K-fold stacking algorithm and the LSTM network. Particularly, compared with the K-fold stacking algorithm, the increment of the F1 score is more than 11% in FFmpeg, Qemu, and the combined projects. Compared with the LSTM network, the F1 score is improved by 0.2012 (29.53% higher) in FFmpeg, and 0.0557 (6.67% higher) in the combined dataset. It proves the effectiveness of extra CNNs in our design to achieve better features. It also can be observed that the performance on the combined dataset is better than that of a single project. This confirms that a large amount of data is indeed a key factor in boosting deep neural networks.

Approach	Precision	Recall	F1
KFS + full word2vec	0.8493	0.8780	0.8634
KFS + partial word2vec	0.8048	0.9142	0.8560
LSTM + full word2vec	0.8299	0.8724	0.8506
AVDS + full word2vec	0.8646	0.9022	**0.8830**

Table 9 Evaluation results on Linux dataset

Approach	Precision	Recall	F1
KFS + full word2vec	0.7986	0.7916	0.7951
KFS + partial word2vec	0.7208	0.7638	0.7417
LSTM + full word2vec	0.8531	0.5672	0.6814
AVDS + full word2vec	0.9012	0.8648	**0.8826**

Table 10 Evaluation results on FFmpeg dataset

Approach	Precision	Recall	F1
KFS + full word2vec	0.7004	0.6807	0.6904
KFS + partial word2vec	0.7972	0.5547	0.6542
LSTM + full word2vec	0.7799	0.7356	0.7571
AVDS + full word2vec	0.8087	0.7407	**0.7732**

Table 11 Evaluation results on Qemu dataset

Approach	Precision	Recall	F1
KFS + full word2vec	0.8153	0.7389	0.7752
KFS + partial word2vec	0.7503	0.6273	0.6833
LSTM + full word2vec	0.8124	0.6707	0.7348
AVDS + full word2vec	0.8641	0.7652	**0.8101**

Table 12 Evaluation results on WireShark dataset

Approach	Precision	Recall	F1
KFS + full word2vec	0.8056	0.7696	0.7872
KFS + partial word2vec	0.8240	0.7253	0.7715
LSTM + full word2vec	0.8253	0.8439	0.8345
AVDS + full word2vec	0.8817	0.8989	**0.8902**

Table 13 Evaluation results on combined C dataset

Other Languages Dataset. Because there is not previous work on Java/JavaScript/Python/Ruby/Go, we evaluate the results with our word2vec models trained under different sets of hyperparameters and compared them with the pre-trained GloVe model. Overall the GloVe has achieved slightly better results than word2vec. Particularly, as shown in Table 14, the F1 score of Twitter GloVe model is 0.85% better than full word2vec model 200 features while Common Crawl Glove model is 1.52% worse than the full word2vec with 200 features. Considering the size of the models, where Common Crawl is 331MB, Twitter is 1.91GB, and word2vec is only 24.6MB, the performance of word2vec is pretty good.

For word2vec model, we notice the performance drops significantly when we increase the *min_words* from 5 to 10 and dimension. The F1 score is increased by 0.0039 when we expand the dimension from 100 to 200, decreased by 0.172 when we further expand the dimension from 200 to 500. This agrees with the earlier Google Test results in section 6.1.

In can also be observed that, unlike the C dataset results, the extra CNNs do not give better features (2.74% worse with CNNs) than LSTM. This shows that the extra CNNs is not effective for a smaller dataset (15,000 commits for other languages compared with 50,000 commits for C).

Approach	**Accuracy**	**Recall**	**Precision**	**F1**
LSTM + full word2vec (100D+ 5 min_words)	0.9051	0.7340	0.7977	0.7684
LSTM + full word2vec (100D + 10 min_words)	0.7902	0.0015	0.1864	0.0030
LSTM + full word2vec (200D + 5 min_words)	0.7902	0.0001	0.0833	0.0003
LSTM + full word2vec (200D + 10 min_words)	0.9018	0.7766	0.7604	**0.7645**
LSTM + full word2vec (500D + 5min_words)	0.8527	0.5106	0.7059	0.5926
AVDS + full word2vec (100D+ 5 min_words)	0.8225	0.3191	0.6593	0.4301
AVDS + full word2vec (200D + 5 min_words)	0.9040	0.7500	0.7833	0.7663
AVDS + GloVe (200D + 6B)	0.9029	0.7872	0.7590	**0.7728**
AVDS + GloVe (Twitter 200D + 27B)	0.9007	0.7287	0.7829	0.7548

Table 14 Evaluation results on combined other languages dataset

7. Conclusion and Future Work

We present an automated deep neural network based vulnerability detection system for Git commit messages. First of all, we conducted a large-scale study of Git commits from multiple open-source projects on GitHub. The vulnerability commits dataset is first-of-its-kind and can be used as a baseline for future research. As a by-product of the data processing step, we developed a web-based triage portal for security admins to label commits in a clean and scalable fashion. Finally we designed and implemented a deep neural network approach to automatically predict the vulnerability status of a commits based on the commit messages. The network has achieved a precision as high as 0.8817, with a recall rate of 0.8989 on the combined C datasets, beating the state-of-the-art K-fold stacking algorithm by 13% in F1 score. For other languages, our neural network also achieved an accuracy of 0.9040 with a F1 score of 0.7663.

Due to time constraint, there are some limitations in our work that could be improved in future work. For example:

(1) In our dataset building processing, the keyword-filtering mechanism relies on the regular expression matching to our human curated dictionary. The approach could accidently filter out some commits that are still security-relevant but do not contain any keywords. We think in future work we could use techniques like Latent Semantic Analysis (LSA) to further improve the accuracy of the filtering process.

(2) The web-based CVE Triage system greatly simplified the labelling work and saved previous time of our security researchers. However the system is tightly coupled with our internal authentication system. In the future, we think of open source the platform and make it open to the community.

References

[1] M. Dowd, J. McDonald and J. Schuh, The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities, Addison-Wesley Professional.

[2] D. Z. Morris, “How Equifax Turned Its Massive Hack Into an Even Worse ‘Dumpster Fire’,” 2017. [Online]. Available: http://fortune.com/2017/09/09/equifax-hack-crisis/. [Accessed 2019].

[3] synk.io, “The State of Open Source Security,” 2017. [Online]. Available: https://snyk.io/stateofossecurity/pdf/The%20State%20of%20Open%20Source.pdf.

[4] “National Vulnerability Database,” [Online]. Available: https://nvd.nist.gov/. [Accessed 2019].

[5] S. E. Ponta, P. Plate, A. Sabetta, M. Bezzi and C.´. Dangremont, “A Manually-Curated Dataset of Fixes to Vulnerabilities of Open-Source Software,” in *Proceedings of the 16th IEEE/ACM International Conference on Mining Software Repositories*, 2019.

[6] “FlawFinder,” [Online]. Available: https://dwheeler.com/flawfinder/.

[7] “Annotation-Assisted Lightweight Static Checking,” [Online]. Available: http://splint.org/.

[8] “PREfast Analysis Tool (Windows CE 5.0),” [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/embedded/ms933794(v=msdn.10).

[9] “Find Bugs in Java Programs,” [Online]. Available: http://findbugs.sourceforge.net/.

[10] “An extensible multilanguage static code analyzer.,” [Online]. Available: https://github.com/pmd/pmd.

[11] “Fortify Static Code Analyzer,” [Online]. Available: https://www.microfocus.com/en-us/products/static-code-analysis-sast/overview.

[12] “Coverity Static Application Security Testing (SAST),” [Online]. Available: https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html.

[13] C. Hollar, K. Herzig and A. Zeller, “Fuzzing with code fragments,” in *Security'12 Proceedings of the 21st USENIX conference on Security symposium*, Bellevue, WA, 2012.

[14] I. Papagiannis, M. Migliavacca and P. Pietzuch, “DMCA PHP Aspis: Using Partial Taint Tracking To Protect Against Injection Attacks,” in *Proceedings of 2nd USENIX conference on Web application development*, 2011.

[15] Y. C. Cho, D. Babić, P. Poosankam, K. Z. Chen, E. X. Wu and D. Song, “MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery,” in *Proceedings of the 20th USENIX conference on Security*, 2011.

[16] C. Cadar, D. Dunbar and D. Engler, “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Program,” in *8th USENIX Symposium on Operating Systems Design and Implementation*.

[17] B. Livshits and T. Zimmermann, “DynaMine: Finding common error patterns by mining software revision histories,” in *Proceedings of the 10th European Software Engineering Conference held Jointly with the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE’05)*, 2005.

[18] H. Perl, S. Dechand, S. Matthew, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl and Y. Acar, “VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code

Audits," in *CCS '15 Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, Colorado, USA, 2015.

[19] M. Doyle and J. Walden, "An empirical study of the evolution of PHP web application security," in *Proceedings of the 3rd International Workshop on Security Measurements and Metrics (MetriSec'11)*, 2011.

[20] Y. Shin, A. Meneely, L. Williams and J. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," in *IEEE Trans. Softw. Eng. 37*, 2011.

[21] Y. Shin and L. Williams, "An initial study on the use of execution complexity metrics as indicators of software vulnerabilities," in *Proceedings of the 7th International Workshop on Software Engineering for Secure Systems (SESS'11)*, 2011.

[22] A. Bosu, C. J. Carver, M. Hafiz, P. Hilley and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM International Symposium on Foundations of Software Engineering (FSE'14)*, 2014.

[23] P. Morrison, K. Herzig, B. Murphy and L. Williams, "Challenges with applying vulnerability prediction models," in *Proceedings of the Symposium and Bootcamp on the Science of Security (HotSoS'15)*, 2015.

[24] A. Meneely, H. Srinivasan, A. Musa, R. A. Tejeda, M. Mokary and B. Spates, "When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Baltimore, MD, USA, 2013.

[25] Y. Shin and L. Williams, "An Empirical Model to Predict Security Vulnerabilities using Code Complexity Metrics," in *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, 2008.

[26] T. Zimmermann, N. Nagappan and L. Williams, "Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista," in *Proceedings of the Third International Conference on Software Testing, Verification and Validation*, Paris, France, 2010.

[27] K. Chae, H. Oh, K. Heo and H. Yang, "Automatically generating features for learning program analysis heuristics for C-like languages," in *Proceedings of the ACM on Programming Languages*, New York, NY, USA, 2017.

[28] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Paderborn, Germany, 2017.

[29] D. R. Greenfeld and A. R. Greenfeld, Two Scoops of Django 1.11: Best Practices for the Django Web Framework, Two Scoops Press, 2017.

[30] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[31] A. Sabetta and M. Bezzi, "A Practical Approach to the Automatic Classification of Security-Relevant Commits," in *Proceedings of 34th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018.

[32] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," 2013.

[33] "gemsim - Word2vec embeddings," [Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html.

[34] R. ŘEHŮŘEK, "Word2vec Tutorial," 2014. [Online]. Available: https://rare-technologies.com/word2vec-tutorial/.

[35] J. Pennington, R. Socher and D. C. Manning, “GloVe: Global Vectors for Word Representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014.

[36] “Convert glove format to word2vec,” [Online]. Available: https://radimrehurek.com/gensim/scripts/glove2word2vec.html. [Accessed 2019].

Appendix A - Keywords for Filtering (by Language)

Keyword
advisory, attack, authenticate, authentication, brute force, bypass, check, clickjack, compromise, constant time, constant-time, corrupt, crack, craft, crash, credential, cross−origin , cross site, cross Site Request Forgery, cross-Site Request Forgery, CVE-, Dan Rosenberg, deadlock, deep recursion, denial of service , denial-of-service, directory traversal, disclosure, divide by 0, divide by zero, divide-by-zero, division by 0, division-by-0, division by zero, dos, double free, endless loop, exhaust, exploit, expose, exposing, exposure, fail, fixes CVE-, forgery, fuzz, general protection fault, GPF, grsecurity, guard, hack, harden, hijack, https://bugzilla, illegal, improper, infinite loop, infinite recursion , info leak, initialize, injection, insecure, invalid, KASAN, leak, limit, lockout, long loop, loop, malicious, malicious, man in the middle, man-in-the-middle, mishandle, MITM, negative, null deref, null-deref, NULL dereference, null function pointer, null pointer dereference, null-ptr, null ptr deref, NVD, off-by-one, OOB, oops, open redirect, oss-security, OSVDB, out of array, out of bound, out-of-bound, overflow, overread, override, overrun, panic, password, PoC, poison, prevent, privesc, privilege, privilege, proof of concept, protect, race, race condition, RCE, ReDoS, remote code execution, replay, sanity check, sanity-check, security, security fix, security issue, security problem, session fixation, snprintf, spoof, syzkaller, traversal, trinity, unauthorized, unauthorized, undefined behavior, undefined behaviour, underflow, unexpected, uninitialise, uninitialize, unrealize, use after free, use-after-free, valid, verification, verifies, verify, violate, violation, vsecurity, vuln, vulnerab, XML External Entity, XSRF, XSS, XXE

Table 15 Keywords for C

Keyword
advisory, attack, auth check, authentication, billion laughs, brute force, Bugzilla, bump, bypass, bytes comparison, certificate, check, cipher, clean up, clickjack, code execution, compromise, constant time, constant-time, corrupt, crack, craft, crafted, crash, credential, cross-origin, cross site, cross Site Request Forgery, cross-Site Request Forgery, CSRF, CVE-, deadlock, decrypt, deep recursion, denial of service, denial-of-service, deprecate, deserial, deseriali, deserialization, directory traversal, disclosure, DoS, DTD, encrypt, endless loop, escap, escape, exhaust, exploit, exposure, external entities, external xml entities, failure, file inclusion, fix, fixes issue, Fix for #, forgery, forward, hack, harden, hash collision, hijack, HMAC, HTTP smuggling, illegal, improper, infinite loop, infinite recursion, inject, injection, insecure, invalid, latest version, leak, lfi, limit, local file inclusion, long loop, malicious, man in the middle, man-in-the-middle, manipulat, marshaller, misconfiguration, mitm, NVD, open redirect, oss-security, OSVDB, overflow, overread, overrid, overrun, OWASP, password, path traversal, permission, poison, pollution, POODLE, prevent, privesc, privilege, privilege escalation, proof of concept, protect, race, race condition, RCE, recursion, ReDoS, reflected file download, remote code execution, replay, RFD, risk, saniti, Sanity check, secret, secure, security, security fix, security issue, security problem, sensitive, serializ, Server Side Request Forgery, session fixation, side channel, signature, smuggling, spoof, SSRF, string comparison, symlink, time-constant, timing channel, traversal, unauthoriz, Undefined behavior, Undefined behaviour, underflow, unmarshall, untrust, unvalid, valid, validation, verifi, verification, verify, violate, violation, vuln, vulnerab, vulnerable, whitelist, XML External Entity, XSRF, XSS, XXE

Table 16 Keywords for Java

Keyword
attack, attacker, brute force, bypass, clickjack, code injection, command injection, Compromise, constant time, cross−origin , cross site, Cross site Request Foregery, cross Site Request Forgery , cross-Site Request Forgery , Cross Site Scripting, CSRF, CVE-, denial of service , directory traversal, dos, escape, forgery, hack, hijack, malicious, man in the middle, man-in-the-middle, MITM, off-by-one, open redirect, oss-security, overflow, PoC, proof of concept, RCE, ReDoS, remote code execution, serializ, session fixation, spoof, sql injection, unauthori[z|s]ed, underflow, unprivileged, vuln, XML External Entity, XSRF, XSS, XXE

Table 17 Keywords for JavaScript/Python/Ruby/Go

Appendix B – Repository List for Commits Crawling

Repository
ownership-plugin, pmd-plugin, nifi, cxf-fediz, core, keycloak, spring-security, openmeetings, beanshell, activemq, orientdb, bc-java, jenkins, jetty.project, javamelody, netty, restlet-framework-java, Resteasy, spring-framework, jackson-dataformat-xml, soapui, geode, neo4j, commons-fileupload, vertx-web, spring-boot, hawtjni, jline2, spring-data-rest, primefaces, java-cas-client, symphony, spring-ldap, lucene-solr, graylog2-server, simplexml, cxf, git-plugin, xstream, tomcat, picketlink-bindings, hadoop, subversion-plugin, infinispan, undertow, deeplearning4j, jackson-rce-via-spel, drools, lz4-java, spring-security-oauth, vert.x, spring-webflow, blynk-server, wink, plexus-archiver, wicket-jquery-ui, gwt, cas, kafka, gerrit-trigger-plugin, favorite-plugin, spring-social, core-1, cassandra, jolokia, pippo, elasticsearch, rdf4j, commons-compress, halo, jboss-seam, role-strategy-plugin, groovy, badge-plugin, ec2-plugin, aws-codepipeline-plugin-for-jenkins, accurev-plugin, UA-Java, umlet, syncope, github-plugin, hbase, zk, ssh-agent-plugin, flex-blazeds, zt-zip, mojarra, grizzly, framework, mojarra, storm, acra, itextpdf, jbpm-wb, jasypt, sentry, saml-plugin, qpid-broker-j, guava, shelve-project-plugin

Table 18 Java Repository List (Top 100 results)

Repository
bottle, node-ecstatic, bleach, awesome_spawn, cocaine, defaults-deep, air-sdk, delayed_job_active_record, vue, aiohttp-session, flask, mem, brace-expansion, settingslogic, war, password, django, python-virtualenv, cs_comments_service, codejail, configuration, jinja2, e2openplugin-OpenWebif, edx-ora, freenas, ceph-deploy, fail2ban, python-glanceclient, zulip, rply, cpython, calibre, ajenti, tornado, attic, cobbler, fedmsg, ansible-container, python-bugzilla, ansible, shutter, notifier, general, ansible-demo-operations, feedparser, message, sonata, base, html5lib-python, nova, paramiko, ansible-for-rubyists, edx-tim, scrapy, Products.SilvaPhotoGallery, libcloud, pyopenssl, mongo-python-driver, pyxtrlock, salt, restkit, cherrypy, pysaml2, urllib3, buildout, transifex-client, requests, Products.SilvaFind, Products.SilvaNews, django-epiceditor, blueman, superlance, buildbot, gns3-gui, swift, node-jquery, supervisor_cache, pcs, Acquisition, python-oauth2, uri-js, fedup, building, portal, initgroups, lxml, Missing, SleekXMPP, DateTime, apt, torbrowser-launcher, mock, MultiMapping, BTrees, AccessControl, jquery-ui, slimerjs-edge, qs, zeroclipboard, html-page

Table 19 JavaScript/Python/Ruby/Go Repository List (Top 100 results)