



UNIVERSITY OF  
THESSALY

UNIVERSITY OF THESSALY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

---

# Generating Datasets of Vulnerable Software Projects Written in Different Programming Languages

---

*Author:*  
Nikitopoulos Georgios

*Supervisor:* Manolis Vavalis,  
Professor, University of Thessaly

A thesis submitted for the degree of  
*MEng Electrical and Computer Engineering*

Volos, October 12, 2020

Approved by the Examination Committee:

Supervisor **Emmanouil Vavalis**  
Professor, Department of Electrical and Computer Engineering,  
University of Thessaly

Member **Spyros Lalis**  
Professor, Department of Electrical and Computer Engineering, Uni-  
versity of Thessaly

Member **Michael Vassilakopoulos**  
Associate Professor, Department of Electrical and Computer Engi-  
neering, University of Thessaly

Date of approval: 5-10-2020



UNIVERSITY OF  
THESSALY

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

---

Συλλογή Έργων Λογισμικού με  
Ευπάθειες για την υποστήριξη Μελετών  
Μηχανικής Μάθησης

---

Συγγραφέας:  
Νικητόπουλος Γεώργιος

Επιβλέπων: Βάβαλης Εμμανουήλ,  
Καθηγητής, Πανεπιστήμιο  
Θεσσαλίας

Διπλωματική εργασία για την απόκτηση του τίτλου του  
Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών

Βόλος, 12 Οκτωβρίου 2020

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων **Βάβαλης Εμμανουήλ**  
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Λάλης Σπύρος**  
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Βασιλακόπουλος Μιχαήλ**  
Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Ημερομηνία έγκρισης: 5-10-2020

## **DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS**

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant  
Nikitopoulos Georgios

**ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ  
ΔΕΟΝΤΟΛΟΓΙΑΣ  
ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ**

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Ο Δηλών  
Νικητόπουλος Γεώργιος

## **Abstract**

Security Defects cost firms millions of dollars in terms of downtime, disruption and confidentiality breaches. The U.S National Institute of Standards and Technology has estimated the cost of faulty software in the U.S in the tens of billions of dollars per year. Such defects and particularly, security related ones can be noticed in the early stages of development through the use of static analysis tools. Conventional static analysis tools rely on detecting patterns in source code based on predefined, manually-set rules. With the emergence of the open source code software community it has become possible to use data-driven techniques in order to discover vulnerabilities. Such techniques employ the use of Machine Learning in order to replace heuristics and increase adaptation in real-life data as well as decrease the cost of manual labor required for the development of conventional static analysis tools. Such models require labeled data-sets that resemble real-life data. Their performance is also closely linked to the quality and quantity of those data-sets. We present a labeled data-set of pieces of vulnerable source code and its patched counterparts which is extracted from Open Source Software repositories. A support data-set of commit messages related to the sources of the source code data-set is also presented and used to create a scalable semi-automatic data acquisition system which mines and labels vulnerable source code from publicly available Open Source Repositories.

## Περίληψη

Τα ελαττώματα ασφαλείας κοστίζουν στις επιχειρήσεις εκατομμύρια δολάρια λόγω διακοπών ή διαταραχών λειτουργίας και παραβιάσεων του απορρήτου. Το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας των Η.Π.Α. εκτιμά το κόστος του ελαττωματικού λογισμικού σε δεκάδες δισεκατομμύρια δολάρια ετησίως εγχώρια. Τέτοια ελαττώματα, ιδιαίτερα αυτά που σχετίζονται με την ασφάλεια, μπορούν να παρατηρηθούν στα πρώτα στάδια της ανάπτυξης λογισμικού μέσω της χρήσης εργαλείων στατικής ανάλυσης. Τα συμβατικά εργαλεία στατικής ανάλυσης βασίζονται στην ανίχνευση μοτίβων στον πηγαίο κώδικα με βάση προκαθορισμένους, χειροκίνητα ορισμένους κανόνες. Με την ανάπτυξη της Κοινότητας Λογισμικού Ανοιχτού Κώδικα έχει καταστεί δυνατή η χρήση τεχνικών που βασίζονται σε δεδομένα προκειμένου να ανακαλυφθούν ευπάθειες. Τέτοιες τεχνικές χρησιμοποιούν τη χρήση Μηχανικής Μάθησης για την αντικατάσταση ευρετικών μεθόδων και την αύξηση της προσαρμογής σε πραγματικά δεδομένα, καθώς και για τη μείωση του κόστους της μη αυτόματης εργασίας που απαιτείται για την ανάπτυξη συμβατικών εργαλείων στατικής ανάλυσης. Τέτοια μοντέλα απαιτούν επισημασμένα σύνολα δεδομένων που μοιάζουν ή προέρχονται από πραγματικά δεδομένα. Η απόδοσή τους συνδέεται επίσης στενά με την ποιότητα και το μέγεθος αυτών των συνόλων δεδομένων. Παρουσιάζουμε ένα επισημασμένο σύνολο δεδομένων από κομμάτια ευπαθή πηγαίου κώδικα και τα διορθωμένα αντίστοιχα μέρη του που εξάγονται από αποθετήρια λογισμικού ανοιχτού κώδικα. Παρουσιάζεται επίσης ένα συμπληρωματικό σύνολο δεδομένων από commit messages που σχετίζονται με τις πηγές του προηγούμενου συνόλου δεδομένων πηγαίου κώδικα και χρησιμοποιείται για τη δημιουργία ενός κλιμακούμενου ημι-αυτόματου συστήματος απόκτησης δεδομένων, το οποίο εξορύσσει και επισημαίνει ευάλωτο πηγαίο κώδικα από δημόσια διαθέσιμα αποθετήρια ανοιχτού κώδικα.



To my lovely parents Stavroula and Nikitas  
Thank you for bearing with me

### **Acknowledgements**

I would like to thank Dr. Dimitris Mitropoulos, Head of GRNET's Reliability Engineering Directorate for essentially co-supervising this thesis. I would also like to express my gratitude to Prof. Panos Louridas for his feedback and constructive criticism and Prof. Elias Houstis for being a catalyst in my development as a researcher. Last but not least, I am forever grateful to all of the unnamed faceless people who contributed willingly or unwillingly towards the completion of this project.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Data Resources . . . . .	5
1.1.1	Relevant Source Code Management . . . . .	5
1.1.2	NVD . . . . .	6
1.2	Related Work . . . . .	7
1.3	Research Contribution . . . . .	7
<b>2</b>	<b>Datasets</b>	<b>9</b>
2.1	Vulnerable Source Code Data-set . . . . .	9
2.2	Commit Message Dataset . . . . .	13
<b>3</b>	<b>Method</b>	<b>16</b>
3.1	Retrieving vulnerable source code referenced in the National Vulnerability Database	16
3.1.1	Parse Module . . . . .	16
3.1.2	Flattening Module . . . . .	16
3.1.3	File Retrieval Module . . . . .	16
3.1.4	File Hierarchy Creation Module . . . . .	16
3.2	Searching the web for possibly vulnerable commits . . . . .	17
3.2.1	Targeted Commit Search Module . . . . .	17
3.2.2	Github Search API Module . . . . .	17
3.2.3	Classification UI . . . . .	17
3.3	Paths . . . . .	17
<b>4</b>	<b>Commit Classification User Interface</b>	<b>19</b>
4.1	Layout . . . . .	19
4.2	Combating information overload with saliency maps . . . . .	20
4.3	Controls . . . . .	20
4.4	Enhancing the Data-set and sentiment classification models . . . . .	20
<b>5</b>	<b>Commit Messages</b>	<b>21</b>
5.1	A preferable commit message . . . . .	21
5.2	A non-preferable commit message . . . . .	21
5.3	Word Frequency . . . . .	21
5.3.1	Stemming . . . . .	22
5.3.2	Lemmatization . . . . .	22
5.3.3	Stop Words . . . . .	22
5.3.4	A Simple Scoring Algorithm . . . . .	22
5.3.5	A more complex scoring algorithm: Random Forest / Bag of Words (BOW)	23
5.4	n-grams . . . . .	24
5.4.1	CountVectorizer and Classic Models . . . . .	24
5.4.2	Convolutional Neural Networks . . . . .	24
5.4.3	Convolutional Neural Network Attention and creating Saliency Maps . . . . .	26
<b>6</b>	<b>Conclusion</b>	<b>28</b>
<b>A</b>	<b>Abbreviations</b>	<b>29</b>
	<b>Bibliography</b>	<b>32</b>

# List of Figures

1.1	Data Driven Feedback Loop . . . . .	8
2.1	File Hierarchy . . . . .	9
2.2	CWE to File Extension Heat-Map . . . . .	10
2.3	Commits by CWE (20 most prevalent ones) . . . . .	11
2.4	Number of files associated with each CWE (20 Top ones) . . . . .	12
2.5	Word Cloud generated by commit messages associated with vulnerability patches .	13
2.6	Distribution of amount of words in each commit for vulnerability patches . . . . .	13
2.7	Word Cloud generated by commit messages not associated with vulnerability patches	14
2.8	Distribution of amount of words in each commit for generic commits . . . . .	14
3.1	Project Architecture Diagram . . . . .	18
4.1	The main menu layout . . . . .	19
4.2	Pop up window displaying a coloured commit message saliency map . . . . .	20
4.3	A Commit Message flagged correctly as a False Positive one . . . . .	20
5.1	Decision Tree . . . . .	23

# List of Tables

2.1	Performance metrics difference of models using $N = 1$ as training data compared to the ones using $N = 5$ . . . . .	15
5.1	Words with significant frequencies. Left: SQL Injection, Right: Cross-Site Scripting. Note: Original words are indicative examples. Multiple words can result to the same stemmed word. . . . .	22
5.2	Performance metrics of classic Machine Learning Models using different vectorizers	24
5.3	Performance metrics of the CNN Model (For the vulnerability patch class) using GloVe embeddings . . . . .	26
5.4	Performance metrics of the CNN Model (For the vulnerability patch class) without the GloVe embeddings . . . . .	26

# Chapter 1

## Introduction

Security defects cost firms millions of dollars in terms of downtime, disruption and confidentiality breaches. [1] Even though it is hard to size the economic implications of such attacks, the National Institute of Standards and Technology has estimated the cost of faulty software at 60 Billion U.S. Dollars per year. [2] In 2011 the Ponemon institute published a benchmark study of 50 U.S. companies. [3] According to this report Cyber crimes can do serious harm to an organization's bottom line as they have found that it costs an average of 5.9 Million USD per year per organization.

Such security defects can be found in the early stages of development through the use of analysis tools such as static analyzers and dynamic analyzers. Static analysis, as opposed to dynamic analysis, does not require the execution of programs, as it operates by analyzing the code of the program in order to detect flaws present in it. Simple static analysis tools like Flawfinder [4] are no more sophisticated than a simple grep or find command.[5] [6] More sophisticated tools use a variety of techniques such as parsing the source code files into an Abstract Syntax Tree (AST) and then analyzing it [7]. Another technique involves using the Control Flow Graph (CFG) of the program in order to symbolically execute the program by replacing input data with symbolic values and then analyzing their use over the CFG paths [8]. All those techniques rely on predefined rule-based detection systems which capture a limited subset of possible errors. With the recent boom of the open source code community it has become possible to use data-driven techniques in order to discover vulnerabilities. [9]

Machine Learning (ML) techniques opt to replace heuristics by finding features automatically. In comparison, rule-based systems are too brittle and cannot handle the diversity of real-life data as well as statistical-based methods can. In classic static analysis rules are empirically derived which is something that requires manual effort, whilst ML based analysis automatically generates rules through training. [10] The disadvantage is that Machine Learning models require labeled data-sets that resemble real-life data in order to train them. The quality and size of those labeled data-sets are closely linked with the performance of the model as well as with its ability to generalize, namely, to have the ability to be effective over new data.

### 1.1 Data Resources

#### 1.1.1 Relevant Source Code Management

Git is a distributed version control system used by many software developers during project development as it helps them to manage source code and enables them to keep every version of the project they have worked on. [11] Open Source Software (OSS) projects have adopted Git and other Distributed Version Control Systems (DVCS) as it allows developers to work collaboratively without a single point of failure, contrary to Centralised Version Control Systems. Git gives each developer a local copy of the full development history and changes are copied from one such repository to another. [12] These changes are imported as added development branches and can be merged in the same way as a locally developed branch. Git has five types of objects one of which is of particular interest to us (the commit):

- A blob (binary large object) is the content of a file. Blobs have no metadata. (A blob's name internally is a hash of its content.)

- The equivalent of a directory in git is the **tree object**. A list of file names is contained, each with some type bits and a reference to a blob or tree object that is that file, symbolic link, or directory's contents. The source tree is described by this object.
- A history is comprised by linking tree objects together with **commit objects**. They contain the name of a tree object (of the top-level source directory), a timestamp, a log message also known as the **commit message**, and the names of zero or more parent commit objects.
- A **tag object** can reference another object and can hold added metadata related to that object. Most commonly, it is used to store a digital signature of a commit object corresponding to a particular release of the data being tracked by Git. Example: v1.2 for version 1.2.
- A **packfile object** is a zlib version compressed of various other objects for compactness and ease of transport over network protocols.[13]

A developer makes changes into the repository by creating commits. When a software vulnerability is patched in an Open Source Software repository, the file changes will be described through one or multiple commits. Usually the developer that wrote the patch will reference his action through the log message (also referred to as commit message). By finding the commit the vulnerability was patched, one can retrieve the vulnerable and non-vulnerable version of a file by viewing the versions of the same file before and after that particular commit.

Github is a platform which provides hosting for software development and version control using Git. It hosts over 100 million repositories many of which are public. This makes Github the largest host of source code in the world to date. [14] It provides an application programming interface (API), which is used extensively in this project, that enables the access of its resources. It also offers the feature of Pull Requests (PR). Pull Requests make it easier for users to collaborate through Github as they provide a method for developers to notify other team members that they have finished the creation of a feature branch (multiple commits) in order to allow its inclusion in the master branch. Pull requests can be tagged in Github by topic which allows the search of security related PRs. Unfortunately, through experimentation with this method of searching commits that fix vulnerabilities, it was discovered that a very small amount of patches are employed by this manner which renders this method of finding vulnerable source code cost ineffective.

### 1.1.2 NVD

The National Vulnerability Database (NVD) is the United States government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics. [15] It provides a list of Common Vulnerabilities and Exposures (CVEs) which are publicly known information security vulnerabilities and exposures. Each CVE contains information description about the issue such as a text description, a Common Weakness Enumeration ID, references and other data such as the severity of the issue etc.

#### Common Weakness Enumeration

Common Weakness Enumeration (CWE) is a community-developed list of common software and hardware weakness types that have security ramifications. [16] Each CWE ID is referring to a type of weakness. Weaknesses in this context can be flaws, vulnerabilities, bugs or other errors. CWEs are more of a graph than a list. There are weakness classes which fan out to more weakness bases which can include weakness variances. For example, the SQL Injection weakness base (CWE-89) is the child of the Improper Neutralization of Special Elements in Data Query Logic (CWE-943) class, which is a sub-class of the Injection Class (CWE-74). This tree structure is allowing researchers to group related weaknesses and subsequently vulnerabilities.

#### References

References are URL links and other information which can point at third party advisories, patches, vulnerability reports, exploits, mailing lists reporting the issue etc. Throughout this project reference Github URLs which contain patches are utilized.

## 1.2 Related Work

The data-set presented is designed to be used for the purposes of training Machine Learning (ML) models upon source code. Many efforts to create such models have been made in past years using a variety of data sources. Scandariato et al. (2014) [17] used a data-set of a single software repository in order to train a Support Vector Machine (SVM) model on a bag-of-words representation. Pang et al. (2015) [18] used the code of four Java-android applications in order to do similar research using also SVMs, but with the inclusion of n-grams. Walden, Stuckman and Scandariato (2014) [19] used three open source projects (Moodle, PHPMyAdmin, Drupal) including 223 vulnerabilities to train a Random Forest model in a bag of words (BOW) representation of the source code in order to compare it with a software metrics based prediction system.

Retrieving data-sets from a limited source of repositories results in the limited variance of the data which could further over-inflate performance results and limit the ability of the model to generalize. The above data-sets used are also very limited in size. To combat the drawbacks of training on a small data-set, researchers use the Software Assurance Reference Dataset (SARD)[20] which includes test suites such as the Static Analysis Tool Exposition (SATE) IV [21] Juliet Test Suite. These data-sets are used by related work involving deep-learning based systems. Prime examples of such work are R Russell et al. (2018)[9], Zhen Li et al. (2018) [22] and Xin Li et al. (2020) [23]. They demonstrate the potential of using Deep Learning to detect vulnerabilities from source code. The problem with using this resource of source code is that the test cases presented in them are synthetic, according to the National Institute of Standards and Technologies. This lowers, of course, the ability of the models to generalize their knowledge over real world data. Russell et al. (2018) [9] include Open Source repository data from Github in the data-set by using a suite of conventional static analysis tools to generate the labels (vulnerable, non vulnerable). This of course results in a quality of data limited by the accuracy of the actual static analysis tools used to label it.

More targeted attempts to gather vulnerability information from open source software publicly available software have been made. Antonios Gkortzis et al. (2018)[24] present a dataset of security vulnerabilities in open-source systems. The versions of the vulnerable repositories as well as data about the severity and type of vulnerability are included as well as a method to clone these versions. No labeling is included of which files or functions are vulnerable. Razvan Raducu et al. (2020) [25] collect vulnerable source code from open source repositories linked to SonarCloud, which is enterprise software that performs conventional static analysis on those repositories, a method of operation closely resembling the effort of R. Russell et al. (2018) [9] and sharing the same quality problems.

## 1.3 Research Contribution

In this thesis, we present a labeled data-set of vulnerable source code and its patched versions which is extracted from real-life data as it is retrieved from Open Source Software repositories. The quality of the dataset is supported by and linked to the quality of the inspection of professionals in the MITRE Corporation and other CVE Numbering Authorities (CNAs) who review requested CVE IDs before they are assigned. A support dataset of labeled commit messages that were included in vulnerability patches is also presented and used to extract features in order to make targeted searches in the web. Repositories involved in CVEs are also kept in order to make searches even more targeted. Implementations of those searching systems were created as well as an assisted manual inspection User Interface tool which create new data for the support data-sets, thus constructing a feedback cycle which allows for the scalability of this data acquisition system into a possibly fully autonomous one. Figure 1.1 shows a graph of how commit message data is processed and fed back into the system.



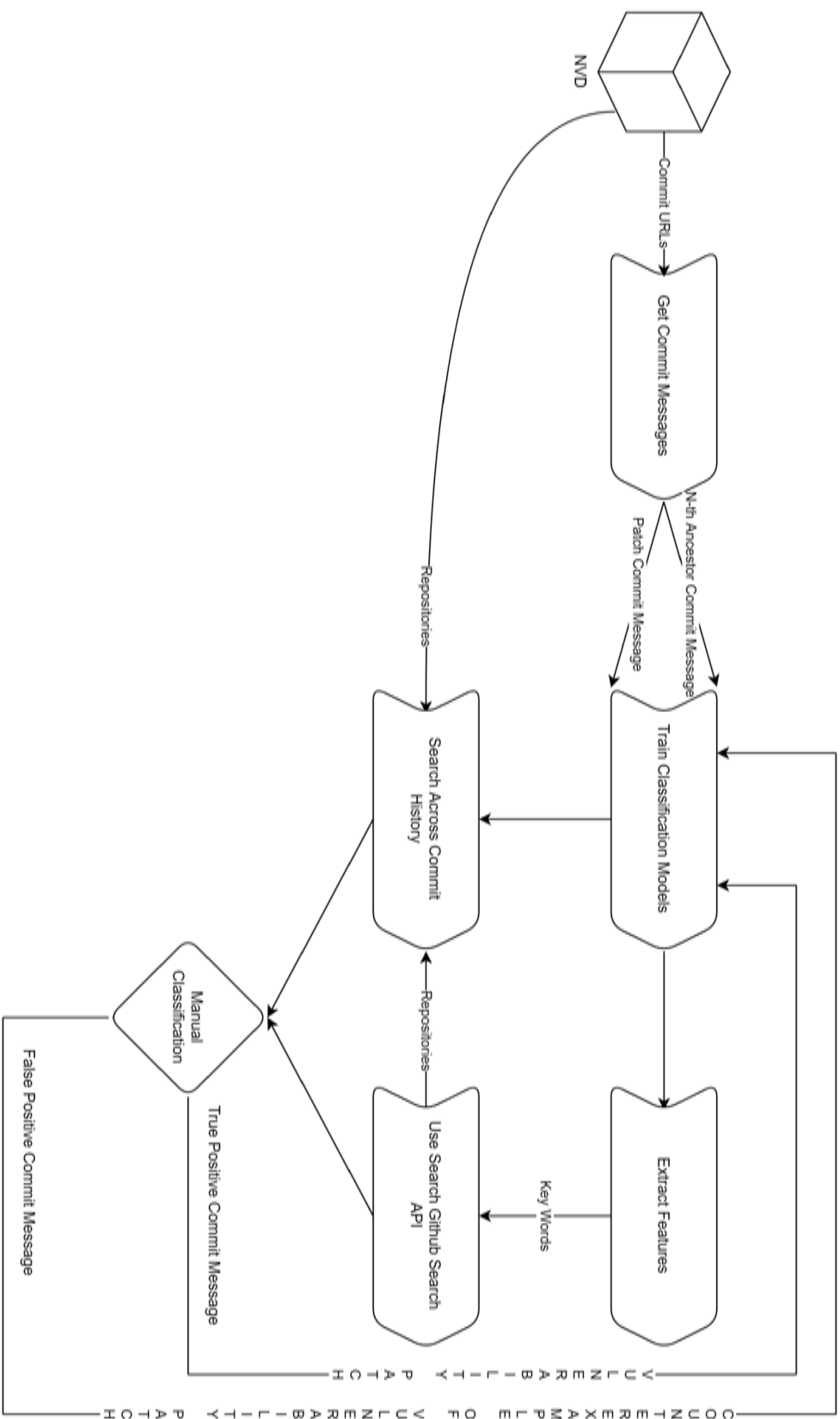


Figure 1.1: Data Driven Feedback Loop

# Chapter 2

## Datasets

The datasets presented in this chapter were generated using the method detailed in Chapter 3. The data contained in them is derived from publicly available open source software repositories. Metadata about the origin of each data point is kept in separate json files which are available in the release of the project.

### 2.1 Vulnerable Source Code Data-set

This data-set contains 911 Megabytes of source code containing overall 32520 files. Exactly half of those files contain vulnerabilities and the other half are the respective patched versions of the vulnerable source code. The data-set is ordered by the type of vulnerability and then grouped by file extension name. This allows researchers to quickly retrieve files of a specific language involving a particular class of security vulnerabilities. The file hierarchy of the data-set is shown in Figure 2.1 where <Filename> is referring to the following naming convention: <good-or-bad>\_<CVE-ID>\_<Number\_of\_file>. The first tag denotes if it is old vulnerable code or the corrected and patched version of the file. The following file extensions are kept in the data-set: c, php, js, py, h,

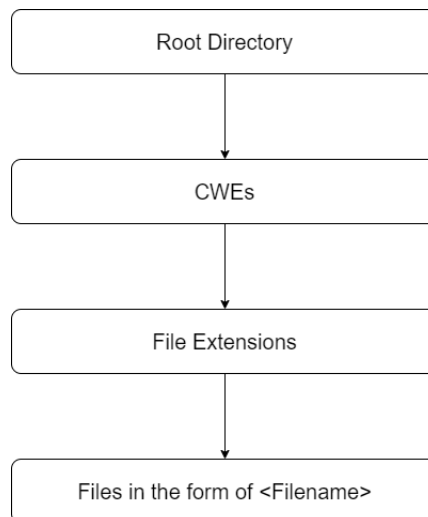


Figure 2.1: File Hierarchy

rb, java, cpp, go, html, xml, tpl, json, cs, cc, pm, sh, phpt, m, inc, scala, cxx, jsp, ctp, jelly, t, htm, scss, tt, as, rs, pl, S, spec, conf, vim, htaccess, hh, lua, coffee, ts, css, phtml, cgi, yml, sql, yaml. The heat-map in Figure 2.2 describes the distribution of each file extension prevalence in respect to its respective weakness. The most prevalent Weakness to File extension pairs are (From left to right, Top to Bottom):

- CWE-119 to .c files. CWE-119 refers to Improper Restriction of Operations within the Bounds of a Memory Buffer which includes one of the most prevalent vulnerabilities in the CVE list, the buffer overflow. Buffer overflows are most prevalent in C and C++ [26] due to

the lack of built-in protection against accessing or overwriting data in any part of memory, its error-prone idioms such as null-terminated strings and its culture that favors performance over correctness.

- CWE-125 to .c files. Out of bounds, also very prevalent in C and C++
- CWE-20 to .c files. Improper Input Validation, a language independent weakness that looks to be most prevalent in C files in this data-set.
- CWE-264 to .c files. Permissions, Privileges, and Access Controls. Another language independent weakness.
- CWE-79 to .js files. JavaScript is the language of the web so it is not at all surprising that Cross-site Scripting vulnerabilities would involve JavaScript.
- CWE-79 to .php files. Another language of the web, php, is expected to be very prone to Cross-Site scripting vulnerabilities.
- CWE-89 to .php files. SQL Injections seem to affect mostly php.

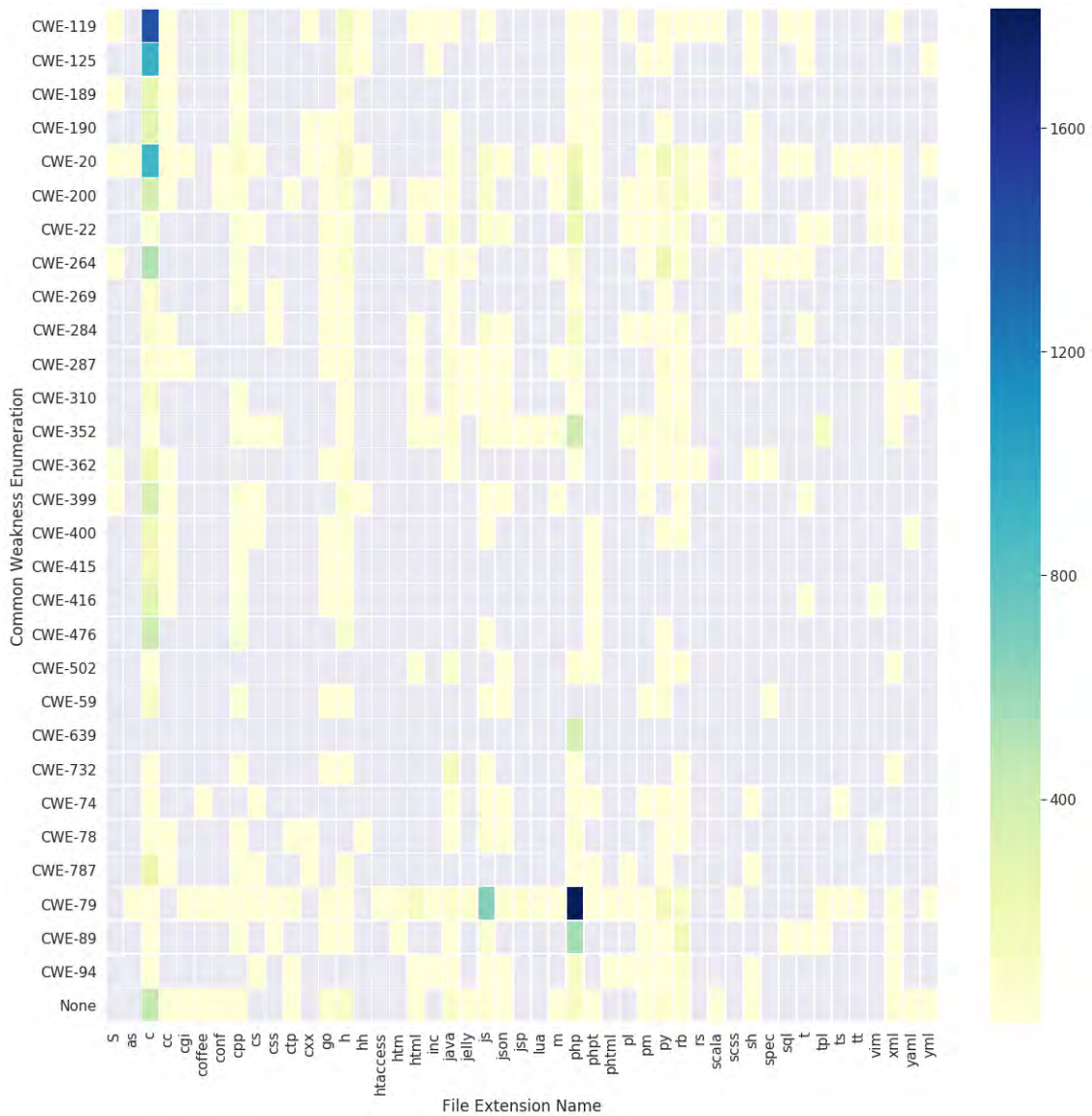


Figure 2.2: CWE to File Extension Heat-Map

One can see the distribution of vulnerable commits per type of vulnerability by looking at Figure 2.3. The five most prevalent vulnerability types being:

- **CWE-79: Cross-Site Scripting Vulnerability.** A web application vulnerability which enables attackers to inject scripts into web-pages which can then be executed by other users viewing them. These vulnerabilities are very prevalent and popular and occur when web applications include untrusted data into their web-pages without escaping special characters or validating the input.
- **CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer:** This is an "umbrella" CWE which includes the classic buffer overflow attack. Buffer overflows are still a very prevalent vulnerability. In the year 2019 alone over a thousand buffer overflow vulnerabilities were documented by CVEDetails [27].
- **CWE-20: Improper Input Validation.** A pretty general type of weakness which can cause an application to crash, use more resources or be exploited.
- **CWE-125: Out of bounds read.** A weakness associated with other such as buffer overflows, integer overflows etc.
- **CWE-200: Exposure of Sensitive Information to an Unauthorized Actor**

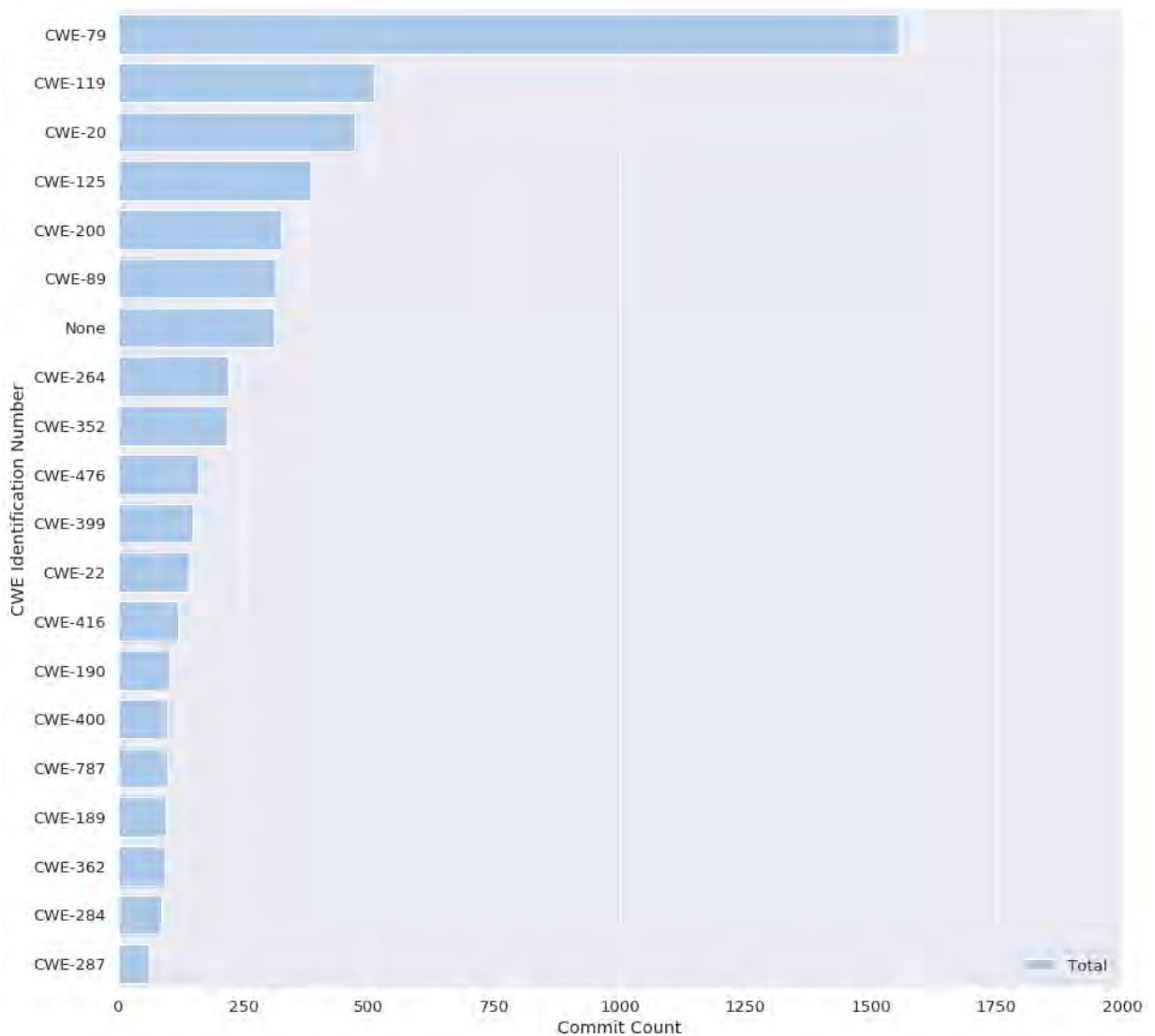


Figure 2.3: Commits by CWE (20 most prevalent ones)

Figure 2.3 is not indicative of the total material one has in order to investigate source code for a particular type of vulnerability. Figure 2.4 displays the total count of files for each type of vulnerability.

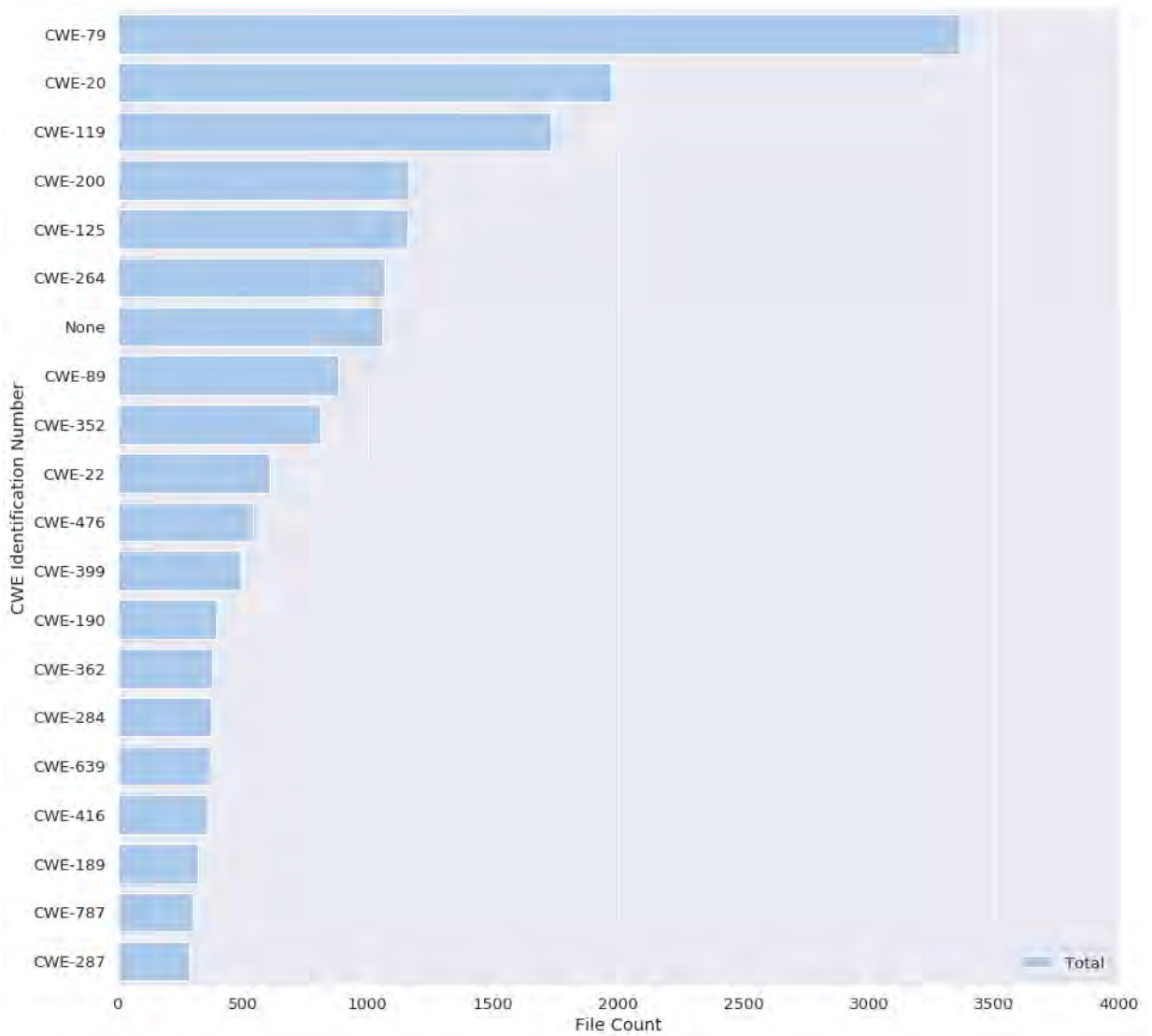


Figure 2.4: Number of files associated with each CWE (20 Top ones)





Machine Learning Model	Precision	Recall	F1	Accuracy
Models Using Glove Embeddings				
None	-8%	-6%	-7%	-6%
Lancaster Stemming	-10%	-1%	-5%	-8%
Lemmatized	-9%	-8%	-8%	-9%
Porter Stemming	-11%	-3%	-7%	-9%
Models Without Glove Embeddings				
None	-12%	-2%	-7%	-8%
Lancaster Stemming	-11%	+6%	-5%	-8%
Lemmatized	-13%	0%	-6%	-9%
Porter Stemming	-9%	-7%	-9%	-10%

Table 2.1: Performance metrics difference of models using  $N = 1$  as training data compared to the ones using  $N = 5$

Of course half of the data-set containing generic commit messages can be replaced for different values of  $N$  in the process of obtaining  $N$ -th ancestors. It was mentioned that low values of  $N$  will corrupt the data labeled as non vulnerability patch commit messages with commit messages from vulnerability patches. This property is demonstrated by training our models in the data-set with  $N = 1$  and  $N = 5$  and observing the difference in performance. Table 2.1 shows how worse models trained with a dataset of depth  $N = 1$  by demonstrating the percentage differences in performance from models trained at  $N = 5$ . Setting the parameter of  $N$  to five is the standard by which the actual performance of the models was evaluated. Tables 5.3 and 5.4 demonstrate these measurements



# Chapter 3

## Method

There are three simple main concepts in the methodology by which we extract vulnerable source code data. The first methodology is collecting all known Github references from the National Vulnerability directory. This allows for the retrieval of reliable data which can then be used to further expand the data-set. The second methodology searches for commits that fix vulnerabilities in repository lists generated by the first methodology using keyword matching inside the commit message. The third methodology is essentially looking in the entirety of Github public repositories for targeted Vulnerability Types (CVEs) using keyword searching through the Github API.

### 3.1 Retrieving vulnerable source code referenced in the National Vulnerability Database

#### 3.1.1 Parse Module

This module requires the obtained National Vulnerability Database Feed json files. The json files contain data about each year's CVE reports. This module parses each file and looks for Github links using string matching in the `reference_data` field. It then associates it with its assigned CVE and creates a comma separated file with the URL and its relevant CVE identification number.

#### 3.1.2 Flattening Module

This module receives the comma separated file created by the Parse Module and looks for URLs that are Pull Requests. Pull Requests contain a multitude of commits each, which are then extracted by querying the page response using an XPath. XPath stands for XML Path Language which can be used to query XML documents as well as HTML documents which have a very similar structure to XML. The separation of this module is crucial to the architecture of the workflow because it serves as an entry-point to pull request links.

#### 3.1.3 File Retrieval Module

This module takes as an input the comma separated file created by the Flattening Module now containing only commits which are CVE labeled. Out of those links the parent commits are obtained (the commit hashes) for each of them and then the parent commit links are constructed. Next, for all of the files changed in the vulnerability fix commit two versions of them are obtained, one from the vulnerability fix commit (the fixed file) and one from the parent commit (the vulnerable file). They then are labeled `good_<commit_id>_<file_id>` and `bad_<commit_id>_<file_id>` respectively. At the same time, it creates a `file_names.list` file containing the original filename (which is important for later filtering) associated with the data-set file named according to the convention described above.

#### 3.1.4 File Hierarchy Creation Module

This module takes as an input the filenames list, the csv created by the Flattening Module and the directory of all the files downloaded by the File Retrieval Module. It then associates each file with

its respective CWE and filename extension and places it in the `/<cwe_id>/<filename_extension>` path under the root directory thus creating the file hierarchy detailed in Chapter 2.1.

## 3.2 Searching the web for possibly vulnerable commits

The U.S. National Vulnerability Database and any other vulnerability database like the Chinese National Vulnerability Database (CNNVD) [29] upload only a limited number of vulnerabilities each year compared to the amount of vulnerable code that is patched every single day by security professionals. The downside of searching the web is the fact that no trusted party is guaranteeing the quality of the data.

### 3.2.1 Targeted Commit Search Module

This module takes as an input a list of repositories that are known to be audited by security professionals. Two options were used on the course of this project as input lists one being a list of the highest CVE rated open source systems created by the project VulnOSS [24] and repository lists generated by the data-set which could target specific repositories that contained vulnerabilities of certain types. This module uses those repositories to search each commit history to find keywords related to types of vulnerabilities inside the commit message data field. The keyword matching is done by comparing Stemmed (Lancaster[8]) words. This method allows for a type of fuzzy string searching. Commits that are found to possibly be vulnerability patches are then scored by Machine Learning models described in paragraphs 5.3.5 to 5.4.2, based on their commit message. The list of the commit urls, scores, CWEs and data regarding activations of the Convolutional Neural Network are then outputted in a quoted comma separated file for further inspection

### 3.2.2 Github Search API Module

This module takes as an input keywords related to a specific type of vulnerability as well as a list of programming languages. It then searches Github for commits related to those keywords. This has to be done iteratively and by date because the Github Search API limits search results to 1000. The process that follows resembles the one followed by the Target Commit Search Module. Commits are scored by the same Machine Learning models described in paragraphs 5.3.5 to 5.4.2, based on their commit message. The list of the commit urls, scores, CWEs and data regarding activations of the Convolutional Neural Network are then outputted in a quoted comma separated file for further inspection. The output files generated by the Github Search API Module and the Targeted Commit Search Module have to be the same in order for Classification UI to process them.

### 3.2.3 Classification UI

The classification UI which is further explained in chapter 4 is allowing a user to distinguish between commits that fix vulnerabilities and false positives. It is important to keep false positives because the machine learning models will have representative data of commits that look like vulnerability commits but really aren't.

## 3.3 Paths

One can use Figure 3.1 as a guideline to get a general idea of the project architecture. To retrieve the baseline data-set the `{1, 2, 3, 12, 14}` path must be followed alongside `{11, 13}`. Then, in order to search the web for possibly vulnerable commits either path `{9, 10, 12}` or `{8, 10, 12}` can be used.

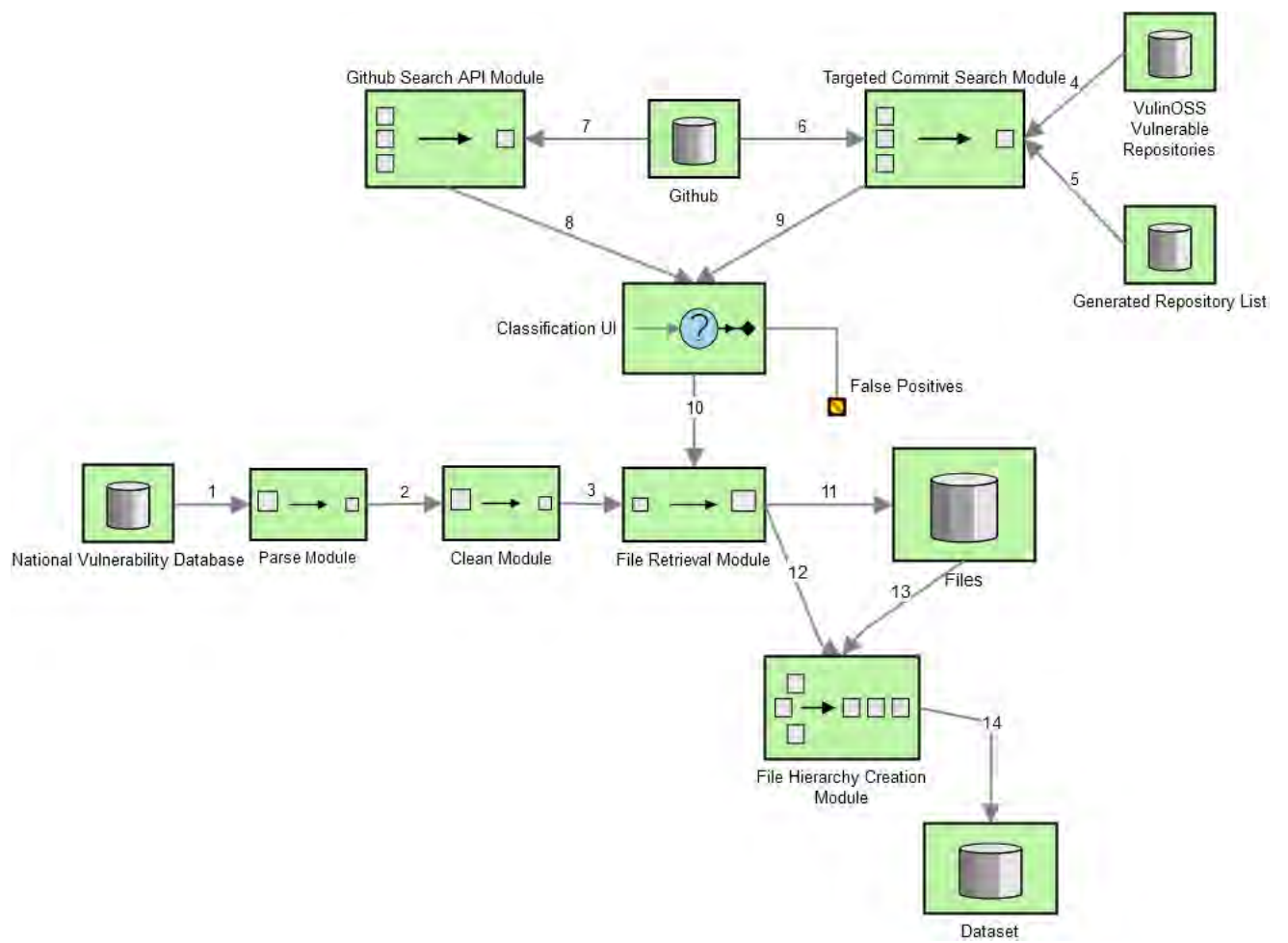


Figure 3.1: Project Architecture Diagram

# Chapter 4

## Commit Classification User Interface

This interface serves the purpose of distinguishing between commits that fix vulnerabilities and ones that do not. It is a terminal application controlled only by the keyboard. This allows the researcher to quickly audit the commit messages and the commits. The `curseXcel` [30] open source library was used to order the commits in rows whilst the columns contain information about each commit. The `ncurses` library [31] is used as the underlying user interface framework.

### 4.1 Layout

The main menu is organised in four columns. Column one contains the URL of the commit. It is important to display this information to a researcher because one can recognise from it the project in which the commit exists. The second column displays the activations of the final layer of neurons of the Convolutional Neural Network that scores the commit message. The third column displays the decision of the Random Forest Model that scores the commit message. The fourth column displays the type of the vulnerability that either the Github Search API Module or the Targeted Commit Search Module detected through fuzzy string matching. Upon inspection the user can display a pop-up window of the commit message to further audit it. It is recommended that this application is used in conjunction with a browser in order to also review the changes in the source code of the files.

Commit Url	Neural Network Decisions (ba..	Random Forest Decision	CWE
https://www.github.com/owncl..	[[0.93818706 0.06181299]]	1	352
https://www.github.com/owncl..	[[0.93760294 0.06239707]]	1	352
https://www.github.com/owncl..	[[0.9771239 0.02287603]]	1	352
https://www.github.com/owncl..	[[0.8144188 0.18558125]]	1	352
https://www.github.com/owncl..	[[0.9343536 0.06564645]]	1	352
https://www.github.com/owncl..	[[0.9037948 0.09620513]]	1	352
https://www.github.com/owncl..	[[0.83404815 0.16595179]]	1	352
https://www.github.com/owncl..	[[0.72879034 0.27120966]]	1	352
https://www.github.com/owncl..	[[0.7961505 0.20384946]]	1	352
https://www.github.com/owncl..	[[0.5505732 0.44942677]]	1	352
https://www.github.com/owncl..	[[0.78684795 0.2131521 ]]	1	79
https://www.github.com/owncl..	[[0.96147084 0.03852911]]	1	352
https://www.github.com/owncl..	[[0.8576233 0.14237675]]	1	352
https://www.github.com/owncl..	[[0.854422 0.14557809]]	1	352
https://www.github.com/owncl..	[[0.9011036 0.09889639]]	1	352
https://www.github.com/owncl..	[[0.7380495 0.26195052]]	1	352
https://www.github.com/owncl..	[[0.92623025 0.0737697 ]]	1	352
https://www.github.com/owncl..	[[0.71809244 0.2819075 ]]	1	352
https://www.github.com/owncl..	[[0.5890563 0.41094375]]	1	352
https://www.github.com/owncl..	[[0.9659475 0.03405244]]	1	352
https://www.github.com/owncl..	[[0.7791432 0.22085677]]	1	352
https://www.github.com/owncl..	[[0.7606889 0.23931104]]	1	352
https://www.github.com/owncl..	[[0.71513397 0.28486603]]	1	352
https://www.github.com/owncl..	[[0.9264688 0.07353117]]	1	352
https://www.github.com/owncl..	[[0.69226044 0.30773956]]	1	352
https://www.github.com/owncl..	[[0.8647157 0.1352843]]	1	352
https://www.github.com/owncl..	[[0.6490119 0.3509881]]	1	79
https://www.github.com/owncl..	[[0.920108 0.07989207]]	1	352
https://www.github.com/owncl..	[[0.78184915 0.21815084]]	1	352
https://www.github.com/owncl..	[[0.7532565 0.24674344]]	1	352
https://www.github.com/owncl..	[[0.712465 0.28753498]]	1	352
https://www.github.com/owncl..	[[0.8473607 0.15263927]]	1	352
https://www.github.com/owncl..	[[0.9691473 0.03085266]]	1	352
https://www.github.com/owncl..	[[0.89129627 0.10870372]]	1	352
https://www.github.com/owncl..	[[0.8661328 0.13386714]]	1	352
https://www.github.com/owncl..	[[0.82995695 0.1700431 ]]	1	352
https://www.github.com/owncl..	[[0.8719753 0.12802471]]	1	352

Figure 4.1: The main menu layout

## 4.2 Combating information overload with saliency maps

One of the most severe problems of perception is information overload. [32]It is very time costly for a researcher to cognitively discard the unimportant parts of the commit message and focus on the phrases or sentences that signify that the candidate commit is indeed a vulnerability patch. Thus, a saliency map can be used to prioritize selection e.g identify the most important information in visual input streams and to use this to improve performance in generating visual data. [33] Saliency maps in this application's context are displayed through gradiently coloured text. Words that are signifying more that indeed the audited commit is a vulnerability patch are brighter red than the ones that aren't. Saliency maps are created by representing the knowledge of the Convolutional Neural Network classifier. Chapter 5.4.3 covers more thoroughly the process by which this is achievable.

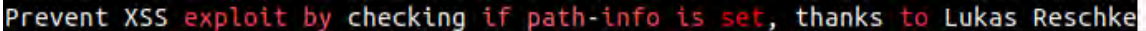


Figure 4.2: Pop up window displaying a coloured commit message saliency map

## 4.3 Controls

The user interface is designed to only be controlled by the keyboard. In a typical QWERTY layout keyboard, the controls for scrolling through the commits and commit messages are grouped on the right hand side whilst the controls for the actual discarding, collecting or auditing of the commits is on the left hand side.

## 4.4 Enhancing the Data-set and sentiment classification models

The list generated by the collected commits and commit messages can be added to the already existing data-set to increase its size and therefore the classifying capabilities of the models used for commit message classification. This allows for the scaling of this project's capabilities. The discarded commits are also kept to have a representation in the data-set of what a False Positive commit message that includes keywords related to vulnerabilities looks like. This allows for further improvement of the efficiency of the sentiment classification models.

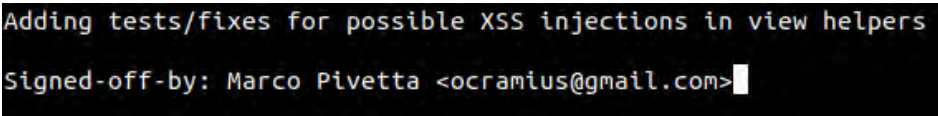


Figure 4.3: A Commit Message flagged correctly as a False Positive one

## Chapter 5

# Commit Messages

Commit messages were mentioned in Chapter 1.1.1. They contain a lot of information about the reason for each commit. In the process of mining commits that fix vulnerabilities one can leverage commit messages in order to determine if a certain commit is indeed a vulnerability fix or not.

### 5.1 A preferable commit message

A real world typical example message that implies a fix of a vulnerability would be commit `b1197e489c330f75c7c11c4b3f382d47c7ffb996` by the `usvn` project with commit message: "Fix JVN 73794686 Cross-site scripting vulnerability". Unfortunately for us most commit messages aren't such textbook telltales of vulnerability fixes. Not every commit message is as short as our aforementioned example. Commit messages often tend to be wordy and contain a lot of information specific to the patch or the problem at hand. Also, not every commit mentions a vulnerability database identification number such as "JVN#73794686", or the words "fix" and "vulnerability" which are very positive indicators that we have a good commit in our hands. Additionally, very conveniently this commit message also contains the type of the vulnerability ("Cross-site scripting") which we know refers to CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')<sup>1</sup>.

### 5.2 A non-preferable commit message

Non preferable commit messages generally constitute messages that have opposite qualities to the preferable example i. e are too lengthy in size, they don't clearly mention that they cover a security issue etc. in the sense that it is hard for a person to verify the vulnerability. But there is a more detrimental kind of commit that needs to be avoided. A typical example of such commit is commit `ae9b6a1b784d9cb45c6eb3273da42440da701996` from the `liferay-portal` repository [34] with commit message: "LRQA-34933 Add test to view no XSS issue in Mobile Device Rules". Such commit looks like a vulnerability patch commit due to it containing the word "XSS" (Cross-site scripting) and the identification number which could constitute vulnerability database identification just like the previous commit. This commit is a test-case addition though and not a vulnerability patch. Such commits are very often test case additions or commits that fix functionality bugs. Also very prevalent are commits from security-educational repositories that actually add vulnerable code into a project. So it is the commits that contain important keywords and that are not actually vulnerability patches the ones that really need to be avoided.

### 5.3 Word Frequency

A common method used in computational linguistics for analyzing text is to construct a list of words accompanied by their respective frequency, where frequency here is referred to how many times they appear in a certain set of texts. In this case it is very useful to group texts that belong in the same weakness (Common Weakness Enumeration) since we can obtain important words specific to each weakness.

---

<sup>1</sup>CWE stands for Common Weakness Enumeration. Refer to Chapter 1.1.2

### 5.3.1 Stemming

Furthermore, words were grouped together in similarity using stemming. Stemming is the process of reducing a word to its root form. This helps group together words that have different suffixes but similar roots such as the words "vulnerabilities" and "vulnerability". The algorithm used for stemming in our case is the Lancaster Stemmer [35] which is a more aggressive alternative to the more popular Porter Stemmer[36]. This was a deliberate choice since we want as much grouping as possible between similar words in this case.

### 5.3.2 Lemmatization

Lemmatization is a method of grouping together the inflected forms of a word to its dictionary form, or lemma. [37] In linguistics inflection refers to the process by which a word is formatted to express different grammatical categories such as tense, case, voice, aspect, person, gender, mood, number, animacy and definiteness. [38] For example, by the process of lemmatization the words "vulnerability" and "vulnerabilities" would both result in the word "vulnerability" because they differ in number. Lemmatization is differing from the process of stemming as the resulting words are actual words of the English language. Thus, it can be seen as a less invasive method of word normalization.

### 5.3.3 Stop Words

Stop words are words that are very common in the English language and therefore are filtered out to obtain more important words. Words such as the word "the" or "is" are not of particular importance to us in this case. The word list used for filtering out stop words is the Natural Language Toolkit's [39] stop word list.

The cases for Cross-Site Scripting and SQL Injection weaknesses are showcased as an example bellow.

Stemmed Word	Original Word	Stemmed Word	Original Word.
fix	fix	xss	xss
sql	sql	fix	fix
inject	injection	us	use
us	use	sec	security
sec	security	issu	issue
vuln	vulnerability	vuln	vulnerability
report	report	php	php
sanit	sanitize	escap	escape
param	paramater	report	report
remov	remove	bug	bug
escap	escape	sanit	sanity
cve	cve	attack	attack
bug	bug	inject	injection

Table 5.1: Words with significant frequencies. Left: SQL Injection, Right: Cross-Site Scripting. Note: Original words are indicative examples. Multiple words can result to the same stemmed word.

This data can be used fairly easily to score text based on individual words. Scoring could be used to distinguish commits that are highly likely to be indeed a vulnerability patch.

### 5.3.4 A Simple Scoring Algorithm

A simple scoring algorithm would be to keep a list of N most frequent stemmed and stop-word filtered terms and count occurrences of them in any original text. So for example, the text "Fix SQL injection" would have a score of 3 based on the SQL Injection word-list and a score of 2 based on the Cross-Site Scripting word-list. One can then find the highest score to even classify the weakness the vulnerability belongs to.

Obviously scoring at the level of words completely ignores the context that they are used in. The presence of the word "vulnerability" could score a commit message in the direction of it being

a vulnerability commit message but in the context of "not a vulnerability" it should have obviously scored against it being a vulnerability patch. In order to maintain the context one needs to examine the text not in an individual word level but in a 'set-of-words' level commonly referred as n-grams.

### 5.3.5 A more complex scoring algorithm: Random Forest / Bag of Words (BOW)

The Bag Of Words (BOW) model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text is represented as an unordered collection of its words, disregarding grammar and even word order. In case of text classification, a word in a document is assigned a weight according to its frequency in the document and frequency in between different documents. Words together with their weights form the bag of words model. [40] In the Bag of Words model, in this particular case, the length of the vector for each piece of text is equal to the number of distinct words to the corpus, i.e the length of the dictionary.

So for example if the dictionary is the set of words  $W = \{ \text{"The", "Sun", "is", "a", "star", "beautiful", "Moon", "satellite"} \}$  then the phrase "The Sun is a star. Sun is beautiful" would be vectorized into [1, 2, 2, 1, 1, 1, 0, 0].

Random Forests are a method of classification invented by Leo Breiman in 2001 [41]. They use a set of decision trees like the one in Figure 5.1. Each tree in this case makes predictions based on the number of occurrences of certain words. Each of these decision tree's predictions will have some variance between them. The Random Forest algorithm combines and averages these predictions to [42] correct the decision trees' habit of overfitting. The name Random Forest derives from the fact that each decision tree takes into account a random subset of the set of all features as well as having access only to a random subset of the training data.

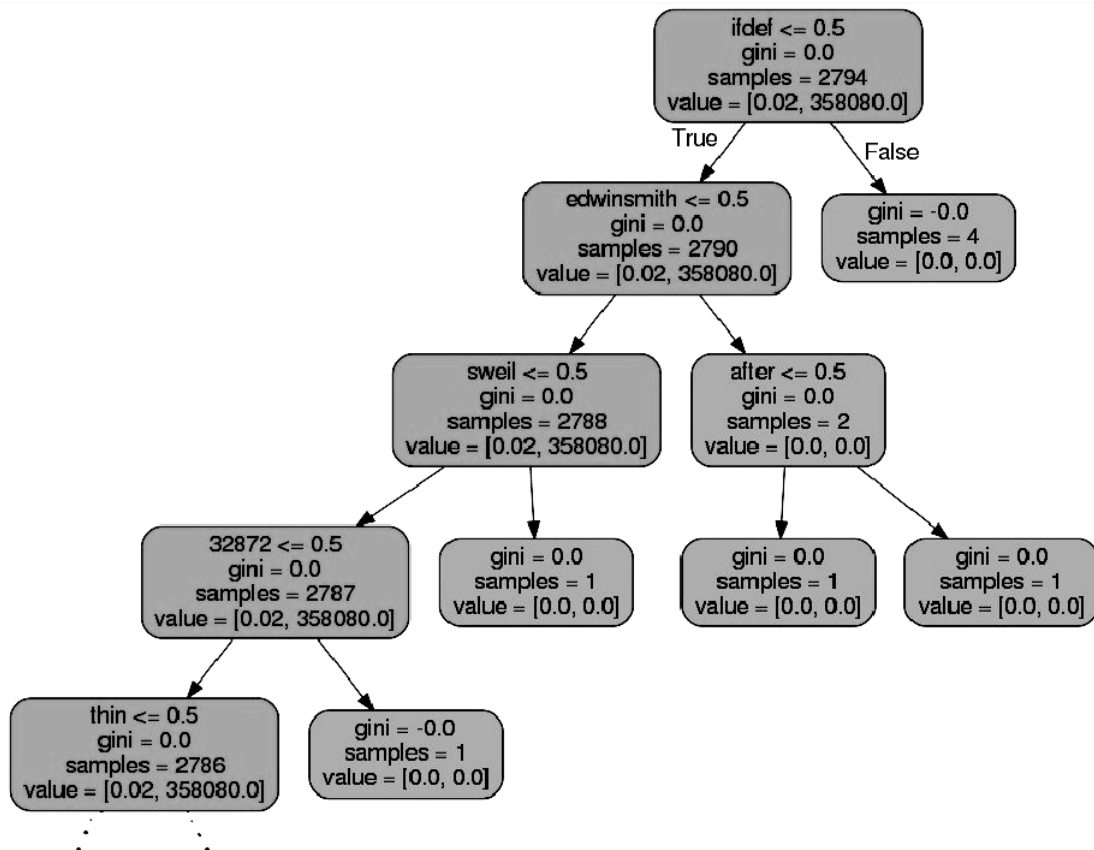


Figure 5.1: Decision Tree



## 5.4 n-grams

In a document classification task, a single key phrase (or an n-gram) can help in determining the topic of the document[43]. N-grams are sequences of N items from a sample of text. The items can be syllables, letters etc. In this application's context n-grams refer to sets of words. [44]. The example mentioned in paragraph 4.3.2 ("not a vulnerability") is a tri-gram because it consists of 3 words. n-grams of words in computational linguistics are also referred to as "shingles" [45]. N-grams essentially allow us to have an idea of the context in which a word is used because the order of the words is kept. This feature of n-grams opposes the bag-of-words model which is an orderless model.

### 5.4.1 CountVectorizer and Classic Models

In a sense it is easy for one to imagine how one can mimic the bag-of-words functionality and extend it to operate in a range of n-grams. The CountVectorizer class in the scikit-learn package [46] does exactly that. One needs to define through the 'analyzer' parameter if the features should be made of word n-grams or character n-grams and then through the 'ngram\_range' parameter the upper and lower boundary of n-values.

One can then apply any popular classifier like the Multinomial Naive Bayes Classifier, the Support Vector Machine (SVM) or as mentioned above and used throughout this thesis project the Random Forest Classifier.

#### Tf-Idf

TF-IDF (Term Frequency times Inverse Document Frequency) is a measure of how many occurrences of a given word is observed (frequency) normalized by its prevalence into the corpus (rarity). It is normally computed as follows. Suppose we have a collection of N documents. Define  $f_{ij}$  to be the frequency (number of occurrences) of term (word) i in document j. Then, define the term frequency  $TF_{ij}$  to be:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \quad (5.1)$$

That is, the term frequency of term i in document j is  $f_{ij}$  normalized by dividing it by the maximum number of occurrences of any term (in this case excluding stop words) in the same document. Thus, the most frequent term in document j gets a TF of 1, and other terms get fractions as their term frequency for this document. The IDF for a term is defined as follows. Suppose term i appears in  $n_i$  of the N documents in the collection. Then  $IDF_i = \log_2(N/n_i)$ . The TF-IDF score for term i in document j is then defined to be  $TF_{ij} \times IDF_i$ . The terms with the highest TF.IDF score are often the terms that best characterize the topic of the document. [47]

Experimentally one can see that Tf-Idf, as shown on Table 5.2, is definitely improving the performance of classic ML Models.

Machine Learning Model	Precision	Recall	F1	Accuracy
Naive Bayes with CountVectorizer	0.77	0.80	0.78	0.77
Naive Bayes with TfIdfVectorizer	0.77	0.83	0.80	0.79
RF with CountVectorizer	0.77	0.81	0.79	0.78
RF with TfIdfVectorizer	0.78	0.79	0.79	0.79

Table 5.2: Performance metrics of classic Machine Learning Models using different vectorizers

### 5.4.2 Convolutional Neural Networks

Another way one can take into account n-grams without the involvement of any grouping of words in the feature vector is the use of Convolutional Neural Networks. Convolutional Neural Networks are usually used in the field of computer vision but have a big use in Natural Language Processing. They take as input word embeddings which are vectors of real numbers mapped directly from words or phrases (n-grams). When those embeddings are derived from the Bag Of Words model (BOW) the model is referred as bow-CNN.

## One-Hot embedding

One-hot vectors are binary vectors. Their dimension in the Natural Language Processing case is as big as the dictionary length. The vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify a word. They are used in Neural Networks to avoid the assumption that words which are next to each other in the dictionary order are similar. One-Hot embedding preserves the word order information unlike the bag-of-words model.

Convolutional Neural Networks utilize convolutions of the input layer (embedding layer). Then subsequent layers apply filters of certain lengths which are then combined by a process called pooling.

Let  $x_i \in \mathbb{R}^k$  be the k-dimensional word vector corresponding with the i-th word in the text. A text of length n is represented as the embedding  $x_{1:n} = x_1 \oplus x_2 \oplus \dots \oplus x_n$  where  $\oplus$  is the concatenation operator. Let  $x_{i:i+j}$  refer to the concatenation of words  $x_i, x_{i+1}, \dots, x_{i+j}$  i.e an j-gram. A convolution operation involves a filter  $w \in \mathbb{R}^{hk}$ , which is applied to a window of h words to produce a new feature. A CNN automatically learns the values of its filters based on the task you want to perform without needing to represent the whole vocabulary. A feature  $c_i$  is generated from a window of words  $x_{i:i+h-1}$  by  $c_i = f(w \cdot x_{i:i+h-1} + b)$ . Here  $b \in \mathbb{R}$  is a bias term and  $f$  is a non-linear function such as the hyperbolic tangent or ReLU. This filter is applied to each possible window of words in the text to produce a feature map  $c = [c_1, c_2, \dots, c_{n-h+1}]$ . [48]

The pooling operation mentioned above in the case of max-pooling is picking the maximum value off of the vector c. In the case of average-pooling the average value is picked. Max-pooling almost in all cases in practice is more effective than average-pooling. Max-Pooling or sub-sampling offers the purpose of picking the most important feature for each map and it also reduces the output dimensionality.

The model uses a multitude of those filters in varying lengths so as to capture multiple features in different n-gram lengths. So in conclusion each filter specializes in a closely related family of n-grams and max-pooling extracts the ones which are relevant. [49]

## Hyperparameters

The filter sizes - sometimes also referred to as kernel\_sizes - are chosen to be 3, 4 and 5. Keeping features for n-grams of length more than five is usually redundant. The number of filters for each kernel size is chosen to be 150. Batch size in the training phase is set to a 100. Additionally, the Adam optimizer is used with a learning rate of 0.001, using the binary cross entropy function.

## GloVe Embeddings

The GloVe model was presented by Jeffrey Pennington, Richard Socher, and Christopher D. Manning. (2014) [50]. GloVe which stands for global vector is a way to represent words as vectors in multidimensional spaces. This model captures semantic and syntactic regularities using vector arithmetic. It is an unsupervised model meaning that it does not require labeled data. In the vectors created, the cosine similarity or the distance between the vectors can be used to find other vectors that represent similar words. Searching for similar keywords to vulnerabilities is very easy using this model's representations. A GloVe model trained on the commit message corpus for example shows the words 'reflected', 'stored', 'persistent', 'possible', 'fixed', 'attacks', 'vulnerability' and 'potential' as the eight most related words to 'xss' which stands for cross-site scripting. Here one can see that a lot of these words are indeed correlated to this type of vulnerability.

A very impressive property of word vector models is the ability to perform arithmetic operations between word vectors to semantically add or subtract meaning. This is used as a sanity check for the purposes of the project so as to verify the efficiency by which the model captures linguistic regularities. According to GloVe model, the following vector operation demonstrates beautifully the semantic relation between the words `sqli`, `injection`, `buffer` and `overflow`.

$$\vec{sqli} - \vec{injection} + \vec{buffer} \approx \vec{overflow} \quad (5.2)$$

The embedding layer of a convolutional neural network essentially maps the word indices into low dimensional vector representations. The GloVe embeddings are used to initialize this layer with the pre-trained values.

## Convolutional Neural Network Performance

Different versions of the commit message data-set were created to find the optimal way to pre-process the data before training or predicting. Two stemmed versions of the data-set were created one using the Porter stemmer and one using the Lancaster one. Additionally, a lemmatized version was created. A version of the original commit message data-set was also tested.

Normalization Algorithm	Precision	Recall	F1	Accuracy
Lancaster Stemming	79%	80%	79%	79%
Lemmatization	79%	82%	80%	79%
None	79%	77%	78%	77%
Porter Stemming	80%	80%	80%	80%

Table 5.3: Performance metrics of the CNN Model (For the vulnerability patch class) using GloVe embeddings

Of course, the model without the pre-trained GloVe embeddings has slightly worse performance, as seen on Table 5.4.

Normalization Algorithm	Precision	Recall	F1	Accuracy
Lancaster Stemming	77%	80%	79%	78%
Lemmatization	80%	77%	78%	78%
None	79%	76%	77%	77%
Porter Stemming	76%	83%	80%	79%

Table 5.4: Performance metrics of the CNN Model (For the vulnerability patch class) without the GloVe embeddings

## Improving Vulnerable Commit retrieval relevance

One can improve the performance of a model by recognizing which statistic is the most important in a certain application. In a data mining setting where the underlying problem is that one can not boil down a lot of candidate commits into the ones that are actually vulnerability patches the statistic of precision is the most crucial one.

$$Precision = \frac{tp}{tp + fp} \tag{5.3}$$

Essentially, when the goal is to remove False Positives the Precision statistic has to be as close to 1.0 as possible. One can sacrifice the increasing of False Negatives in order to improve this statistic. This, in the context of neural networks can be done by moving the decision threshold. The neural network outputs a probability of  $p = [0, 1] \in \mathfrak{R}$  that the commit belongs in the vulnerability patch class. By default, commits that output a probability of  $p > 0.5$  are selected to be classified as vulnerability patches. One can increase the number of False Negatives and reduce the number of False Positives by increasing this threshold. Indicatively, one can increase this threshold to 0.7 or 0.8 to achieve precision rates upwards of 85% to 90%. This is roughly a 5% to 10% improvement from the precision rates displayed in Table 5.3

### 5.4.3 Convolutional Neural Network Attention and creating Saliency Maps

Due to the huge potential of deep learning, interpreting neural networks has become one of the most critical research directions. Deep learning works as a black box model in the sense that although it performs quite well in practice, it is difficult to explain its underlying mechanism and understand its behaviors. [51] In this section the method used to interpret the attention of the Neural Network model (Simonyan et. al. (2013)) [52] is outlined and the adaptation of it in the context of Natural Language Processing is explained.

## Vanilla Gradients for Images

This technique computes a class saliency map, specific to a given input and class. Given an image  $I_0$  (with  $m$  rows and  $n$  columns) and a class  $c$ , the class saliency map  $M \in \mathfrak{R}^{m \times n}$  is computed as follows. First, the derivative  $w$  is found by back-propagation.

$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0} \quad (5.4)$$

Where  $w$  is the derivative of  $S_c$  (the scoring function) with respect to the image  $I$  at the point (image)  $I_0$ . After that, the saliency map is obtained by rearranging the elements of the vector  $w$  as  $M_{ij} = |w_{h(i,j)}|$ , where  $h(i, j)$  is the index of the element of  $w$ , corresponding to the image pixel in the  $i$ -th row and  $j$ -th column.

## NLP Adaptation

Adapting this algorithm for images to work in a Convolutional Neural Network that takes as an input sequences of indexes of words is fairly straight forward. Following the same procedure as above one obtains a saliency map  $M_{ij} \in \mathfrak{R}^{m \times n}$  where  $m$  are the embedding dimensions and  $n$  is the maximum number of words kept in a sequence. The saliency map is then reduced to a map of  $M_j \in \mathfrak{R}^n$  where:

$$M_j = \sum_{k=1}^m M_{k,j} = |w_{h(k,j)}| \quad (5.5)$$

So here we are summing the absolutes of the gradients of the embedding dimension together to reduce the dimensions of the saliency map.  $M_j$  is essentially a measure of which word  $j$  needs to be changed the least in order to affect the prediction the most.

The values of  $M_j$  are then normalized to the range of  $[0, 1]$  by calculating which helps in further scaling the values to the  $[0, 255]$  range in order to be used for text colouring purposes.

$$\hat{M}_j = \frac{M_j - \min(M_j)}{\max(M_j) - \min(M_j)} \quad (5.6)$$

## Chapter 6

# Conclusion

We have presented a data-set of vulnerable source code and its respective patched/secure counterparts retrieved from open source software repository git commits. We have also obtained the commit messages involved in those commits and trained models to distinguish vulnerability patch related commit messages. A technique to interpret the decision of one of those models is performed. A complete semi-automatic system of vulnerable source code discovery integrating those models is also implemented and tested. To date this project as far as we know is the first complete solution to retrieving vulnerable source code from publicly available open source repositories without requiring the occurrence of any reference from a vulnerability database. It is also the first system which employs machine learning to discover commit messages involved in vulnerability patches.

### **Future Work**

We hope that this system will be deployed in the future in a web application which will allow multiple people at the same time to update its knowledge in order to hopefully scale it into full automaticity. This will not only increase the size of the available vulnerable source code that we have but also increase the ability to retrieve more of it. This subsequently will allow static analysis based on Machine Learning tools to be used not only experimentally but in large scale and in the workplace to improve the early detection of security weaknesses.

# Appendix A

## Abbreviations

API: Application Programming Interface  
AST: Abstract Syntax Tree  
BLOB: Binary Large Object  
BOW: Bag Of Words  
BOF: Buffer Overflow  
CFG: Control Flow Graph  
CNA: CVE Numbering Authority  
CNN: Convolutional Neural Network  
CNNVD: China National Vulnerability Database  
CVE: Common Vulnerability and Exposures  
CWE: Common Weakness Enumeration  
DVCS: Distributed Version Control System  
GloVe: Global Vector  
IR: Information Retrieval  
ML: Machine Learning  
NLP: Natural Language Processing  
NVD: National Vulnerability Database  
OSS: Open Source Software  
PR: Pull Request  
SARD: Software Assurance Reference Dataset  
SATE: Static Analysis Tool Exposition  
SCAP: Security Content Automation Protocol  
SQL: Structured Query Language  
SVM: Support Vector Machine  
TF-IDF: Term Frequency–Inverse Document Frequency  
UI: User Interface  
XML: Extensible Markup Language  
XPath: XML Path Language  
XSS: Cross Site Scripting

# Bibliography

- [1] Rahul Telang and Sunil Wattal. An empirical analysis of the impact of software vulnerability announcements on firm stock price. *Software Engineering, IEEE Transactions on*, 33:544–557, 09 2007.
- [2] Tassef and Gregory. The economic impacts of inadequate infrastructure for software testing. 05 2002.
- [3] Second annual cost of cyber crime study. [https://www.ponemon.org/local/upload/file/2011\\_2nd\\_Annual\\_Cost\\_of\\_Cyber\\_Crime\\_Study%20.pdf](https://www.ponemon.org/local/upload/file/2011_2nd_Annual_Cost_of_Cyber_Crime_Study%20.pdf). Accessed: 2020-8-13.
- [4] David A. Wheeler. Flawfinder. <https://dwheeler.com/flawfinder>. Accessed: 2020-8-13.
- [5] David A. Wheeler. How does flawfinder work? [https://dwheeler.com/flawfinder/#how\\_work](https://dwheeler.com/flawfinder/#how_work). Accessed: 2020-8-13.
- [6] Dejan Baca, Kai Petersen, Bengt Carlsson, and Lars Lundberg. Static code analysis to detect software security vulnerabilities - does experience matter? pages 804–810, 01 2009.
- [7] Andrew Walker, Michael Coffey, Pavel Tisnovsky, and Tom Černý. *On Limitations of Modern Static Analysis Tools*, pages 577–586. 01 2020.
- [8] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, July 1976.
- [9] Rebecca L. Russell, Louis Kim, Lei H. Hamilton, Tomo Lazovich, Jacob A. Harer, Onur Ozdemir, Paul M. Ellingwood, and Marc W. McConley. Automated Vulnerability Detection in Source Code Using Deep Representation Learning. *arXiv e-prints*, page arXiv:1807.04320, July 2018.
- [10] Hao Zhang, Danfeng Yao, Naren Ramakrishnan, and Zhibin Zhang. Causality reasoning about network events for detecting stealthy malware activities. *Computers & Security*, 58:38, 01 2016.
- [11] Kathryn D. Scopatz, Anthony; Huff. *Effective Computation in Physics*. O’Reilly Media, Inc, 2015.
- [12] Distributed git - distributed workflows. <https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows>. Accessed: 2020-8-13.
- [13] Git internals - git objects. <https://git-scm.com/book/en/v2/Git-Internals-Git-Objects>. Accessed: 2020-8-13.
- [14] Khari Johnson. Github passes 100 million repositories. <https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/>, 2018. Accessed: 2020-8-13.
- [15] NIST. Nvd main page. <https://nvd.nist.gov/>. Accessed: 2020-8-13.
- [16] The MITRE Corporation. Overview - what is cwe? <https://cwe.mitre.org/about/index.html>. Accessed: 2020-8-13.
- [17] R. Scandariato, J. Walden, A. Hovsepian, and W. Joosen. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, 2014.

- [18] Y. Pang, X. Xue, and A. S. Namin. Predicting vulnerable software components through n-gram analysis and statistical feature selection. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 543–548, 2015.
- [19] James Walden, Jeff Stuckman, and Riccardo Scandariato. Predicting vulnerable components: Software metrics vs text mining. pages 23–33, 11 2014.
- [20] NIST. Nist software assurance reference dataset project. <https://samate.nist.gov/SRD/>. Accessed: 2020-8-13.
- [21] Paul E. Black Vadim Okun, Aurelien Delaitre. Report on the static analysis tool exposition (sate) iv. <http://dx.doi.org/10.6028/NIST.SP.500-297>. Accessed: 2020-8-13.
- [22] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. *arXiv e-prints*, page arXiv:1801.01681, January 2018.
- [23] Xin Li, Lu Wang, Yang Xin, Yixian Yang, and Yuling Chen. Automated vulnerability detection in source code using minimum intermediate representation learning. *Applied Sciences*, 10:1692, 03 2020.
- [24] Antonios Gkortzis, Dimitris Mitropoulos, and Diomidis Spinellis. Vulinoss: A dataset of security vulnerabilities in open-source systems. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR '18*, page 18–21, New York, NY, USA, 2018. Association for Computing Machinery.
- [25] Razvan Raducu, Gonzalo Esteban, Francisco Lera, and Camino Fernández. Collecting vulnerable source code from open-source repositories for dataset generation. *Applied Sciences*, 10:1270, 02 2020.
- [26] C. Cowan, F. Wagle, Calton Pu, S. Beattie, and J. Walpole. Buffer overflows: attacks and defenses for the vulnerability of the decade. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 119–129 vol.2, 2000.
- [27] Serkan Özkan. CVE details website. <https://www.cvedetails.com/>. Accessed: 2020-8-13.
- [28] Andreas Mueller. Word cloud generator in python.
- [29] Chinese national vulnerability database. <http://www.cnnvd.org.cn/>. Accessed: 2020-8-13.
- [30] coffeeandscripts.github.io. curseexcel. <https://github.com/coffeeandscripts/curseXcel>. Accessed: 2020-8-13.
- [31] GNU Project. ncurses library. <https://invisible-mirror.net/archives/ncurses/>. Accessed: 2020-8-13.
- [32] E. Niebur. Saliency map. *Scholarpedia*, 2(8):2675, 2007. revision #147400.
- [33] Derrick Parkhurst, Klinton Law, and Ernst Niebur. Parkhurst d, law k, niebur e. modeling the role of salience in the allocation of overt visual attention. *vision res* 42: 107-123. *Vision research*, 42:107–23, 02 2002.
- [34] The Liferay Development Team. liferay-portal repository. <https://github.com/liferay/liferay-portal>. Accessed: 2020-8-13.
- [35] Chris D. Paice. Another stemmer. *SIGIR Forum*, 24(3):56–61, November 1990.
- [36] M.F. Porter. An algorithm for suffix stripping. *Program*, 14:130–137, 07 2006.
- [37] Collins English Dictionary. "lemmatize".
- [38] Jonathan Owens. Case and proto-arabic, part i. *Bulletin of the School of Oriental and African Studies*, 61:51 – 73, 02 1998.



- [39] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, page 63–70, USA, 2002. Association for Computational Linguistics.
- [40] Soumya K and Shibily Joseph. Text classification by augmenting bag of words (bow) representation with co-occurrence feature. *IOSR Journal of Computer Engineering*, 16:34–38, 01 2014.
- [41] L Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001.
- [42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning. *Aug, Springer*, 1, 01 2001.
- [43] Rie Johnson and Tong Zhang. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1412.1058, December 2014.
- [44] Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 02 2008.
- [45] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157 – 1166, 1997. Papers from the Sixth International World Wide Web Conference.
- [46] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.
- [47] Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman. *Mining of Massive Datasets*. 01 2014.
- [48] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *arXiv e-prints*, page arXiv:1408.5882, August 2014.
- [49] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding Convolutional Neural Networks for Text Classification. *arXiv e-prints*, page arXiv:1809.08037, September 2018.
- [50] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [51] Fenglei Fan, Jinjun Xiong, and Ge Wang. On Interpretability of Artificial Neural Networks. *arXiv e-prints*, page arXiv:2001.02522, January 2020.
- [52] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv e-prints*, page arXiv:1312.6034, December 2013.