Discovering Useful Patterns In Relationships Between Bugs And Vulnerabilities Using Mining Techniques

Maryam Javed

Discovering Useful Patterns In Relationships Between Bugs and Vulnerabilities Using Mining Techniques

By

Maryam Javed

Supervised by

Dr. Onaiza Maqbool

Department of Computer Science

Quaid-i-Azam University

Islamabad, Pakistan

August, 2018

Discovering Useful Patterns In Relationships Between Bugs and Vulnerabilities Using Mining Techniques

By

Maryam Javed

A Dissertation Submitted in Partial Fulfillment for the

Degree of

MASTER OF PHILOSOPHY

IN

COMPUTER SCIENCE

Department of Computer Science

Quaid-i-Azam University

Islamabad, Pakistan

August, 2018

Dedicated to

My loving parents

CERTIFICATE

“Discovering Useful Patterns In Relationships Between Bugs and Vulnerabilities using Mining Techniques”

By

Maryam Javed

A DISSERTATION SUBMITTED IN THE PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF THE

MASTER OF PHILOSOPHY IN COMPUTER SCIENCE

We accept this dissertation as conforming to the required standards

1. _____
   Dr. Onaiza Maqbool
   Assistant Professor
   Incharge
   Computer Science Department

2. _____
   Dr. Onaiza Maqbool
   Assistant Professor
   Supervisor
   Computer Science Department

3. _____
   Dr.
   Assistant Professor
   External Supervisor
   Computer Science Department

Dated: _____

Department of Computer Science

Quaid-i-Azam University, Islamabad, Pakistan

Declaration

I hereby declare that this dissertation is the presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly with due reference to the literature and acknowledgment of collaborative research and discussions.

This work was done under the guidance of Dr. Onaiza Maqbool, Department of Computer Sciences, Quaid-i-Azam University, Islamabad (Pakistan).

Date: August 31, 2018

Maryam Javed

Abstract

With every passing day, the world is becoming more and more dependent on software for everyday tasks. Due to increase in usage, software quality and software security has always been a keen interest of researchers. Engineering a system which is secure, best in design and that fulfills the customer's requirements completely has always been a dream of software engineers. To fulfill this dream, a lot of research is conducted and new techniques are introduced everyday.

Despite the increased focus on making systems that have high quality, software is most likely to have some bugs in it. More people depending on systems for their every day tasks means more data being stored on software which is making software more vulnerable. Everyday millions of hackers try to breach security of software, putting data of millions of customers at stake which can cause loss of millions, can compromise company's position in market and if real time systems are hacked, lives can also be put in danger. Researchers are studying every aspect to make software secure and also to increase software quality because once a software is dispatched, if any bug occurs or any flaw is found it puts the company name at stake.

One recent interest of the researchers has been studying different attributes together to find correlation between them. The findings of the research can be used for beneficial purposes. Software bugs are error, fault or failures which can be caused by some misunderstood requirement, design issue or coding mistake that effects software quality whereas software vulnerabilities can be defined as some weakness or flaw left in the system that could be used to breach security of a system. Software bugs and software vulnerabilities are conceptually different but in history, some bugs were seen to be the cause of major vulnerabilities. In this dissertation,we have studied the relationship between software bugs and software Vulnerabilities. We used Association Rule Mining to find co-occurrences of bugs and vulnerabilities in the Google Chromium Project. We studied the two-sided relationship between bugs and vulnerabilities Bug-> Vulnerabilities and Vulnerability-> Bug. We also analyzed the co-occurrences for the patterns that could be used to improve software security. We analyzed the attributes of Bug and Vulnerabilities Bug and Vulnerability Types, Bug and Vulnerability Scores and Bug and Vulnerability Summaries to

find some patterns in the co-occurrences. We attained 35% of the bugs were found to be co-occurring with vulnerabilities whereas 46% of the vulnerabilities were found to be co-occurring with bugs. We also found some patterns that could be used by google chromium team to improve quality and security of the system.

Acknowledgments

My journey in the University started with a dream coming true, dream that I had been seeing since many years. I would say getting here was rather easy than staying here, it was tough for a mediocre like me. Last three years had alot ups and downs for me, I am still thankful to many people for their never ending support that led me finally complete my thesis.

I would first like to thank my parents for emotional and financial support they had been providing me all my life. Many time, I had done wrong and many times I disappointed them but they never lost their hope in me. I would also like to thank all my siblings for being a pillar of support throughout my life and for being always unconditionally available for me. I can proudly say that I have the best family anyone can have.

I had many emotional up and downs in University, I remember in 1st semester when I was going through a rough phase in a subject, and was quite disheartened about it. I wanted to give up, who didnt let me give up was Dr. Onaiza Maqbool. I still remember from the time I met her and till now , she has always been my inspiration. She was the reason many times I wanted to leave University and I couldn't. I always wanted to make her proud but I always messed it up. I will try to make you proud one day and I will try to adapt everything you used to ask me because I know you were always right. I would also like to thank my professors Dr. Khalid Saleem, Dr. Mudassar Azam Sindu, Dr. Shoaib Karim, Dr Ghazanfar and Dr Akmal Khattak for all the knowledge and guidance they provided me throughout degree. I would like to thank Shabeer Uncle for keeping the environment in the lab calm and quite.

Staying in hostel was my first experience, I was badly disheartened when I came to hostel, I still remember when mama left me in hostel, I had tears. I remember Mam Hina (My hostel Warden), she gave me the much needed hug. In last three years, She made hostel home for me, every time I was sick, or emotional or home sick, she was there. I remember when I used to come back from hostel, Anabia used to hug me so tight calling me "Bajjo" and Aunty was like a mother I had in hostel, holding her hand I could feel comfort of feeling Amma around. Thank you for staying , You people were another family I had and the family who will always stay with me.

I would like to thank one of the best friends I had in my whole life time "Anjuman Ara" I cannot thank you enough for always being there for me, for not judging me and for doing everything you could for me. You are one big blessing that I cannot thank Allah enough for. Also, I would like to thank Aqsa Akbar for helping me throughout the thesis with her inspirational talks and for being always available whenever I needed her.

Saba, Sana, Beenish,Madiha, and Warda, Thanks for making my hostel life memorable. I will remember our adventures, late night talks and all the fun we had together. Thank you Aroosa for being the best room mate anyone could have.

Last but not least I would like to thank Anum Zahra, Rabia Mazhar, Maryam Imtiaz , Shehroz Asmat ,Muhammad Aurangzaib, Jay Kumar, Farooq Zaman, Qamar Munir and Aqib Rehman for the time they invested.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 2011 Marc Andreessen in an article said “Software is eating the world”, he might have not realized that his words would come true soon (AVNER, 2017). Now-a-days use of software is essential to keep up with this fast paced world. Software is being used perform simple to complex tasks whether they are transactions, communication, medical facilities or helplines. Software Engineers have always taken keen interest in improving software quality and software security. Providing customers a system that suits their requirements and to make a secure and non- vulnerable system has been a dream of many software engineers. To make this dream come true, software trends are changing rapidly. Different methodologies and techniques are being introduced to support the idea of providing error free and non-vulnerable software.

No matter how much time and effort is spent to improve the software quality, software is most likely to have bugs. Tracing and removing bugs is a continual process throughout the lifespan of software. The process of tracing and removing bugs (formally known as Testing and Debugging respectively) assures that software has no (or minimum number of) bugs. Software quality is a major concern of software developers these days. With increased complexity, it is difficult to ensure an error-free software.

Software Testing is the process of finding errors or bugs or failures in a system (Myers, 2004). It is used to ensure that program/ software is performing how it is supposed to. Software testing is performed to assure quality and reliability of the system. We don’t test a program only to check whether that program is working, we test a program against requirements, also, we assume that there are some errors/bugs in the system which we have to identify and remove. We also test the system to identify loopholes, loopholes that a hacker can use to harm the system, or get access to it or to perform any action that breaches system security.

Software Security is another major concern of the software industry today. The increased growth

of software industry has also caused the industry of cybercrimes to expand which resulted in industries putting more focus on software security. Every year a lot of security updates are made to software in response to some security breaches that are made or in response to the security sensitive parts of the software found by the software management or security management team. Some companies like Google hire an ethical hacker who continuously attacks their own system to find its weaknesses before the attacker finds these weaknesses and exploits them. One of harsh realities that software engineers have to face is that the pace in which software engineers create more and more secure systems, in the same pace attackers make strategies to break those security walls. But for attacker to attack a system, there has to be some weakness in the system which is exploitable. For example for a thief to enter a house, there has to be some weakness in the security system of the house; either locks are easily breakable, some door is left opened by mistake or some other weakness. Without these weaknesses neither can a thief enter a house nor can an attacker can attack the system. These weaknesses that lead the attacker to attack the system are known as vulnerabilities. Vulnerabilities are already present in the system in the form of some flaw or error. Software Security is the ability of software to function correctly even if it's under a malicious attack or an idea implemented to protect the system against any hacking attacks (Mcgraw, 2004).

Now we are going to explain a few terms that are going to be used repeatedly in thesis.

1.1 Bugs

A software bug is an error, fault or failure in the system (Myers, 2004). Systems can have bugs due to misunderstood requirements, design issues or some coding mistake. Bugs are reported by the users or developers after the system is released, every bug report is stored in a database called bug repository. Every bug report defines some attributes of the bug *e. g*. Bug ID, Bug Priority, Bug Type, Bug Summary.

1.1.1 Types Of Bugs

There are different types of bugs, we are going discuss a few of which are found common by most of the software vendors (Li et al., 2006).

Bug Type	Cause	Effect	Example
Memory bugs	These bugs occur when corrupted memory locations are used.	They can affect other memory locations during program execution and can corrupt them which can also lead to whole program or system failure.	Stack overflow
Semantic Bugs	These are caused by inconsistency in the design requirements given by the customer.	It can result in missing features, missing cases, wrong control flow and many other errors.	Use of a non-initialized variable: int i; i++; // the variable i is not initialized
Security Bugs	They are caused when developer leaves some weakness or loophole in the system.	The affects vary it can be as simple as attacker reading all the data to attacker destroying the whole system.	HeartBleed Bug allows stealing the information protected, under normal conditions
GUI Bugs	These bugs can be caused because of Semantic Bugs, when requirements are misunderstood, graphical user interface is also compromised.	Even if according to the designer Graphical User Interface is perfect, but if it doesn't meet requirements provided by the customer, it's a total failure. In other cases, system might not be having some important modules or some functionalities might be missing.	Misalignment, broken images, wrong sized graphic elements, inconsistent colors on a link, button or menu

Table 1.1: Software Bug Types

Bug Type	Cause	Effect	Example
Concurrency Bugs	These bugs are caused when multi-threading environment is used, also they occur in case of data race, deadlock and synchronization.	It can result in system hangs and system crashes.	Deadlocks, livelocks, starvation, and thrashing

1.1.2 Bug Repositories

Every software receives a set of problems and issues, these problems or issues are stored in a database. For bugs, this data base is normally known as Bug Repository. Every software has its own bug repository which not only contains all the issues a software faces but also detailed information regarding these issues. Bug repositories are used to collect and manage all the information provided by the user as well as information provided by the debugger (Kanwal & Maqbool, 2012). Bug repositories are used to study the patterns of the previous reported and resolved bugs and these patterns could later be used to resolve the new reported bugs quickly. Since a lot of research is being carried out in the field of software quality, these repositories play a vital role in the lives of researchers (Xuan, 2012).

1.2 Vulnerabilities

A software vulnerability can be defined as an error, weakness or a flaw that is most likely to be exploited by an attacker to achieve his goal of breaching the security of the system (Jimenez, Mammar, & Cavalli, 2009). Vulnerability can also be defined as the consequence of an error. When we allow the system to function beyond it's requirements, it may result in increased chances of it being misused. An example as simple as if we divide any number by 0, it results in crash, which can be exploited by the attacker if he discovers it. It is also defined as a type of software bug which has security consequences. According to research, vulnerabilities remain in the system for two years on average before they are identified and fixed. In worst case, software Vulnerabilities can destroy the system completely.

1.2.1 Types of Vulnerabilities

There are different types of vulnerabilities which are common in software(Munaiah, Camilo, Wigham, Meneely, & Nagappan, 2017a).

Vulnerability Type	Cause	Effect	Example
DOS	This type of vulnerability occurs when there's some bottleneck in the software which triggers the CPU usage, memory leaks, disk i/o, database calls.	It can make the whole system collapse and it causes strange errors to make the user feel that system performance is under stress.	CVE-2018-0976: an attacker connects to the target system using Remote Desktop Protocol and sends specially crafted requests.
Overflow	It occurs when attackers send a large amount of data of the application and then adds some malicious attack at the end of data.	Information stored in the stack can be overwritten.	CVE-2017-7269: Buffer overflow allows remote attackers to execute arbitrary code via a long header beginning
Code Execution	Some website weak in security provides the user with server side files that they can modify, code can also be injected in the scripting language of software.	Attackers manipulate the web application output or content by executing some code on the server side.	CVE-2018-5007: Adobe Flash Player 30.0.0.113 and earlier versions have a Type Confusion vulnerability. Successful exploitation could lead to arbitrary code execution in the context of the current user.

Table 1.2: Software Vulnerability Types

Vulnerability Type	**Cause**	**Effect**	**Example**
Memory Corruption	It occurs when the memory is changed without an explicit assignment. Due to some programming mistake content of memory locations are modified outside memory limit.	Content placed in memory can be accessed and altered.	CVE-2018-9465: In a function of a file, there is a possible memory corruption due to a use after free. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation.
SQL Injections	It occurs due to programmatic fault, when programmer neglects to escape strings properly in SQL Queries.	Attacker can delete data from database, he can also alter data or simply copy data and use it against organization.	CVE-2018-5993: SQL Injection exists in the Aist through 2.0 component for Joomla! via the id parameter in a view=showvacancy request.

Table 1.2: Software Vulnerability Types

Vulnerability Type	**Cause**	**Effect**	**Example**
XSS	XSS vulnerabilities occur normally on the web pages, user is knowingly directed to some web page that has XSS vulnerability, once user lands the page he is attacked by the malicious user	In the worst case, one click on the wrong page can delete all the data of the user, give control of the user's system to the attacker.	CVE-2015-3429: This Cross-site scripting (XSS) vulnerability allows remote attackers to inject arbitrary web script or HTML via a fragment identifier.
Directory Traversal	It is caused by user supplied unvalidated input	It can allow the attacker to access restricted directories of software and also lets the attacker execute malicious commands.	CVE-2007-2285: Directory traversal vulnerability allows remote attackers to read arbitrary files via a .. (dot dot) in the feed parameter.

Table 1.2: Software Vulnerability Types

Vulnerability Type	Cause	Effect	Example
Http Response Splitting	It can occur if a malicious user inject their own CRLF sequence in HTTP stream, CRLF refers to the Carriage Return and Line Feed sequence of special characters. Many internet protocols including HTTP used these character as the end of line marker.	It can allow the attacker to gain control over the content of HTTP response.	CVE-2006-6699: Multiple CRLF injection vulnerabilities allow remote attackers to inject arbitrary HTTP headers and conduct HTTP response splitting attacks via CRLF sequences.
Bypass	It occurs when and if system administrator makes some entry point which enables the user to access the system without going through security clearance procedures.	Attacker can get access to the system, the effect is based on what kind of access he got, which external route attacker has discovered and the route leads to which door. Sometimes it only allows the attacker to read information whereas in other cases it can allow data manipulation and data destruction as well.	CVE-2012-3365: The SQLite functionality in PHP before 5.3.15 allows remote attackers to bypass the open_basedir protection mechanism via unspecified vectors.

1.2.2 Vulnerability Repositories

Just like bugs, vulnerabilities are also stored in a database which is called vulnerability repository. Unlike Bug repositories, every software doesn't have separate vulnerability repositories. Every vulnerability that occurs in any software is stored in common database of vulnerability for all the software.

Every vulnerability report comprises of some attributes relating that vulnerability some of them are CVE ID(Common Vulnerability and Exposures), CVSS(Common Vulnerability Scoring System) Score, CVE Summary. National Vulnerability Database(NVD) and Common Vulnerability and Exposure details(CVEDetails) are examples of such repositories (Özkan, 2012).

1.3 Commit Data for Bugs and Vulnerabilities

Commit Data record changes made to the system. When a bug or vulnerability or any kind of fault occurs in the system, system is evolved through small changes in response. These changes are known as commits. Commit data or commit files are the files that are changed while fixing a bug or vulnerability. Every Bug and vulnerability report consists of these commit files as a feature (Alali, Kagdi, & Maletic, 2008).

1.4 Bug-vulnerability relationship

As mentioned earlier vulnerability is a special type of bug which has security consequences. Bugs and vulnerabilities are conceptually different, a bug is an error that may have occurred due to coding error or requirement misunderstanding whereas vulnerability is a security flaw or misuse or abuse of a functionality. Vulnerabilities are hidden functionalities that let the attacker use system beyond the expectations of the developer. It is possible that the hidden functionalities that are causing vulnerabilities to occur are caused by some bug. We are not saying that every vulnerability can be caused by some hidden functionality, it can also be the case that a vulnerability might have allowed the attacker to place bugs in the system as well.

The empirical study of the relationship between bugs and vulnerabilities will not only tell us about the relationship strength but also may reveal some hidden patterns that can be used to improve software security or software security and software quality both. If there exists strong relationship between bugs and vulnerabilities, we can further study whether bugs are causing vulnerabilities or vulnerabilities are causing bugs to occur.

1.5 Motivation

In 2014, iPhone manufacturing company Apple discovered a vulnerability in their system, the vulnerability was registered as CVE-2014-1266 also informally introduced as “goto fail”. As suggested by

the name the vulnerability caused by a single 'goto' statement which was misplaced in the code made all the users vulnerable. The command was used in the code which included OSX and IOS encryption module. The module was supposed to verify an iPhone when it connects to an encrypted site over SSL. But instead of using one GOTO statement two GOTO statements were used. The first one belonged to the code whereas the second one was a typing mistake which diverted the program execution and acted like a bypass stent, it didn't let the module apply encryption, the encryption code was made dead by this additional GOTO statement which made communication of millions of the apple users go vulnerable. Gotofail was a coding mistake, a bug that caused a vulnerability to occur (Greenberg, 2014). Another vulnerability, CVE-2016-10087 was found in May 2016, null-pointer-dereference bug caused it. The bug allowed a malicious user to execute DOS attack by exploiting the bug (AVNER, 2017). In 2017, another vulnerability occurred in jquery in a module where input of a variable was low cased *e.g.* "A" was converted into "a". Whenever that attribute was given Boolean value it created an exception which caused infinite recursion because Boolean values can not be low cased. This small bug caused a vulnerability known as WS-2017-0195 to occur. Due to infinite recursion, system's functionality was broken and it caused Denial of Service(DOS) vulnerability (Abbot, 2017). There are more such vulnerabilities that were seen in history that were caused by some coding mistake that led the attacker to gain access to the system.

Research on relationship between bugs and vulnerabilities is new, not a lot of research has been conducted on this topic, we have found two research papers in which this topic was being explored. In the first research paper by Fonseca, José and Vieira, vulnerabilities-bug relationship was exploited, achieved results of the research was not enough to state that software faults or software bug were responsible for software vulnerabilities (Fonseca & Vieira, 2008). In the second paper by Munaiah, Nuthan and Camil (Munaiah et al., 2017a), we know that software quality can be translated into software security means the issues that arise due to some coding mistake or due to some missed requirement can actually create some loophole in the system that could be manipulated by the attacker. They exploited Bug-Vulnerability relationship in detail and gave verdict that although there exists a relationship but the relationship is not that strong that it can be further explored.

1.6 Objective of Research

The objective of this research is not only to study bug-vulnerability relationship but also discovering useful and hidden patterns that can be further used to improve software security or software quality. In

previous researches, researchers have studied the one sided relationship between bug and vulnerability i. e. bug-> vulnerability. They have exploited relationship in a way where bugs can cause vulnerabilities to occur. We intend to study two sided relationship. We intend to study whether a vulnerability can cause some bugs to occur as well. We also intend to study the relationship by analyzing corresponding attributes of bugs and vulnerability to find any useful pattern that can lead us to improve software quality or software security. We will analyze the relationship using mining techniques, the major technique that we are going to find the co-occurrence of bugs and vulnerabilities is Association Rule Mining. Further, to analyze these relationship, we will use different statistical and mining techniques like inter dependability among attributes, Term Frequency.

1.7 Problem Statement

Relationship between Software Security and Software Quality needs to be exploited to understand the nature of relationship. Studying that relationship can be benificial to find the hidden, useful hidden patterns which can be used to improve Software Quality and Software Software Security.

1.8 Research Questions

1.8.1 Are Commit Files fixed for Bugs or Vulnerabilities likely to be fixed for future Bugs or Vulnerabilities?

Commit Files are the base of the research, Our whole research relies on the ratio of relationship bugs and vulnerabilities posses. For finding the ratio, we check how many bugs and vulnerabilities co-occur in a file. It will become the base of our research, it will provide us data of the co-occurred bugs and vulnerabilities which we will use to exploit the relationship among them.

1.8.2 If a file has one vulnerability, then how likely is it to have another vulnerability in it? Do files with more bugs have more vulnerability?

Some research states that bugs and vulnerability behave in different ways, they say that if a file has the most bugs in it, it's more likely to have more bugs in the same file again. But if a vulnerability occurs in a file, that file is not likely to have any more vulnerability. We want to verify the results. We are trying to find some hidden patterns that could provide us some interesting facts about the relationship, it's possible that in a file a bug and vulnerability occurred and they both were having high priority and

high CVSS Score which could lead us into believing that a high priority bug will always cause a high CVSS Score vulnerability.

1.8.3 Does Security bugs only translate into the vulnerabilities? Are vulnerabilities more likely to be found in high priority bugs?

We are also going to exploit the relationship in terms of attributes of the bugs and vulnerabilities. One aspect that is to be explored is bugs that are related to security will be the only ones that will become vulnerability and will all the bugs relating to security becomes vulnerabilities or not. We exploited bug and vulnerability attributes to find any kind of patterns. Here, we will study if software vulnerabilities are likely to be found in the bugs with high priority.

1.8.4 Are associated bugs increasing the count of vulnerabilities in files? And are associated vulnerabilities increasing the count of Bugs in Files?

We wish to analyze the count of vulnerability in the files with zero bugs to analyze if count of vulnerabilities is effected by the count of bugs in files. In the same way, we also wish to analyze the count of bugs in the files with zero vulnerabilities if count of bugs is effected by the count of vulnerabilities in files

1.9 Thesis Organization

In this chapter, we introduced our research topic and provided research question that we will be answering in our thesis. In Chapter 2, we will be discussing the work to provide base to our research, the work that has been done in this field before. In the Chapter 3, we are going to discuss the methodology that we used to study bugs and vulnerabilities, their relations and pattern that will unveil new facts about them. Chapter 4 describes the experimental setup of the thesis, how we conducted our experiments. Chapter 5 will provide some results of the study, It will also discuss the contribution of the study. In Chapter 6, we will conclude our study and will tell some future work that can be done in the same field.

Chapter 2

Related Work

Software has become a mandatory part of our lives, alot of research has been carried on to improve software quality and software security over time. In this chapter, we are going to discuss previous related literature on bug localization, vulnerability localization and on the the relationship between bugs and vulnerability.

2.1 Bugs

Alot of effort has been put over the time to localize bugs and resolve them. We are going to discuss a few techniques that were proposed for bug localization. In a paper of 1990 (Shahmehri, Kamkar, & Fritzson, 1990), authors presented a semi automated bug localization method which was generalized version of algorithmic debugging. The technique was applicable to all the procedural languages. This debugger provided interactive debugging facility where error could be localized semi-automatically.

In another paper, Liu, Yan and Fei in 2005 (C. Liu, Yan, Fei, Han, & Midkiff, 2005) proposed a new statistical model-based approach known as SOBER which is used to localize software bugs without any knowledge of semantics of program. SOBER models divergence of predicate evaluations between correct and incorrect executions. Rao and Kak in their research have compared five generic text models *i.e*. Unimodel(UM), The Vector Space Model(VSM), the Latent Semantic Analysis Model(LSA), the Latent Dirichlet Allocation Model (LDA) and Cluster Based Document model(CBDM). They studied these models to locate the files that were relevant to bug reports. They studied the models on iBugs dataset and found that UM and VSM were performing better than others for finding the relevant files that were used to fix bugs (Rao & Kak, 2011).

In (Shokripour, Anvik, Kasirun, & Zamani, 2015), authors aimed to improve automatic bug as-

signment by using Time TF-IDF. They have considered the recency of the developer to determine the expertise of the developer in resolving the bug. This technique showed immense improvement in the field of bug assignment. Yan, Zhang and others (Yan, Zhang, Yang, Xu, & Kymer, 2016)have proposed a component recommender by using the proposed Discriminative Probability Latent Semantic Analysis Model which initializes the word distribution for different topics. It trains the model by using historic data of the component assignment in Bug reports which helps software developers to recommend components when a new bug occurs in the system.

In 2015 (Zhang, Wang, & Wang, 2016), Zhang and Wang proposed an approach called K-nearest-neighbours search and heterogeneous proximity which was used to improve automatic bug report assignment. When a bug is reported, KSAP assigns the bug report to the developers using two phase procedure. In the first phase, similar bugs that were resolved in history are found using K-nearest-neighbours method. In the second phase, they created a rank list of the developers who have dealt with similar kind of bugs. The technique helped bug localization by assigning the bug to the developer that had already worked on similar kind of bug. Results of the technique was found better than the others being used for this.

In another paper (Banerjee et al., 2017), authors have proposed an automated triager that can automatically recognize whether the problem is original or duplicate, if the problem is duplicate it provides the list of the duplicated bugs. They used 24 document similarity measures and associated summary statistics to create this fully automated triager.

Le1 T., Lo D. proposed a new framework for fault localization, known as Savant that is a learning-to-rank strategy using likely invariant diffs and suspiciousness scores as feature. It outputs a rank list of the method according to their suspiciousness scores with the method having the most suspiciousness score at the top. Savant is a four step procedure which starts with method clustering and test case selection, invariant mining, feature extraction and method ranking. After these steps are performed, a short ranked list of buggy methods is produced. They used 357 real life bugs from 5 different programs and evaluated Savant on Defect4J benchmark. They claimed that Savant was able to successfully localize 63. 03, 101. 72, 122 bugs on average within the top 1, top 3 and top 5 listed methods. They validated the results of their framework by comparing it with 10 SBFL techniques and have proved that the results of Savant were far better than any of fault localization techniques (Le, Lo, Goues, & Grunske, 2016).

2.2 Vulnerabilities

Software security vulnerabilities are one of the critical issues in the realm of computer security. Due to their potential high severity impacts, many different approaches have been proposed in the past decades to mitigate the damages of software vulnerabilities. Machine-learning and data-mining techniques are also among the many approaches to address this issue. Here, we are going to discuss a few techniques that were used to localize vulnerabilities and resolve them.

Neuhaus and Zimmermann explored the reason for occurrence of vulnerabilities in the system. They have presented an empirical evidence that vulnerabilities correlated with features of the components. They created a tool named as "vulture" which mined the existing databases of vulnerabilities and also the version archives to map the past vulnerabilities to the components in which they occurred. This provided them the data that could be studies to check which components were making software vulnerable. They used Mozilla vulnerability history as the data set and also stated that they found that components that had vulnerabilities in past were not likely to have more vulnerabilities in future. They also stated that features of the components are good predictors of the vulnerabilities. They also created a predictor using suitable set of features for vulnerabilities that could be applied before any component was fully implemented and it could predict whether the component is likely to have any vulnerability or not. They also claimed that their proposed tool was very fast and it could examine a complex project like Mozilla in half an hour and it identifies the 50% of the vulnerable components (Neuhaus, Zimmermann, Holler, & Zeller, 2007).

Authors studied the components that cause the vulnerabilities to occur in the system. They created a predictive model that could idenitify which components of software have high security risks. They used security related static tool warnings, code churn, size and faults identified by the manual inspections as the input variables to the tool and were validated against the vulnerabilities found in past. These results are obtained early while developing the system that allows the developer to know the threats early so that he can have time to redesign or test the most vulnerable software components. They evaluated their model on large cisco software system and claimed that 75. 6 % of the components that were found vulnerable are on the top of list of predicted vulnerable components (Gegick, Rotella, & Williams, 2009).

Jimenez studied vulnerabilities in detail, they state that although vulnerabilities are not a new topic, but still developers don't have idea how to deal them. Vulnerability cause graphs can b used to teach programmers how to avoid vulnerabilities. Also, source code should be inspected to guarantee zero

vulnerabilities during construction phase. Number of tools are also available to detect vulnerabilities, some are based on static techniques while others are based on dynamic. These tools can be used after the software has been developed to ensure that no vulnerability exists (Jimenez et al., 2009).

Gegick studied component proneness for attacks by creating a rank list of the components who are more likely to have vulnerabilities in them. Security Risk management efforts can be made better by using the proposed approach of component rank list. They created a predictive model that identifies the most risky component in software in terms of security. They implemented this model in the early stages of software Development Life Cycle. They used non security factors like Code Churn, SLOC and previous manual inspections of the components to evaluate the component security proneness. They implemented the model on Cisco Software System and found that 75. 6% of the vulnerable components are in top 18. 6% of the components that were predicted by the predictive model proposed by them (Gegick et al., 2009).

Shin discussed the techniques that could be used to prevent and detect vulnerabilities before a new release is made. They performed case studies on Mozilla Firefox Browser and Red Hat Enterprise Linux Kernal, they studied that whether matrices obtained early in Software Development Life Cycle can find vulnerable components and locations in the system. The matrices that they studied were complexity, code churns and developer activity matrices. Results showed that 70. 8% vulnerabilities can be known by selecting 10. 9% files of Mozilla Firefox. Whereas 68. 8% vulnerabilities can be known by selecting 13% files (Jimenez et al., 2009).

In another paper, Bozorgi and Saul state that software vulnerabilities have become big threat to the software industry, so in order to address every vulnerability disclosure, enterprises ranked the vulnerabilities and triaged the vulnerabilities with highest severity first. They stated that the current scoring system available were not accurate because they were based on ad-hoc approaches and are unlikely to produce a robust ranking model for vulnerability. They have created a complementary approach for vulnerability assessment using tools for data mining and machine learning. They have proposed a classifier that predicted that whether a vulnerability is exploitable or not and if it is exploitable how soon it can be exploited. Their classifier operated on the text fields, time stamps, cross-references, and other entries in existing vulnerability disclosure reports which were used to extract dimensional feature vectors. They claimed that their classifier predicted better than the currently used classifier CVSS in NVD (national vulnerability and exposure database) because they have used the machine learning approaches to train their classifier (Bozorgi, Saul, Savage, & Voelker, 2010).

Liu and Shi have studied the software vulnerability techniques including static analysis, fuzzing,

penetration testing. They have discussed the technique, research status and related problems. They have talked in detail about the advantages and disadvantages of each technique. According to the authors, Static analysis can diagnose most the bugs in the development phase whereas fuzzing takes place after the software has been developed but it discovers the most vulnerabilities. VDMs used already discovered vulnerabilities and assesses what threats are faced by the target system, it provides three separate techniques for design and implementation. Penetration testing can find no tool can expose, it used social engineering factors into consideration. Not only they have discussed the current use of these techniques but also future directions of each of the technique (B. Liu, Shi, Cai, & Li, 2012).

Meneely evaluated the effectiveness of the two techniques on vulnerability detection that were effective on bug detection i. e Code Reviews and Validity of Linus' laws. They evaluated the statement "many eyes make all the bug shallow" on vulnerabilities. Vulnerabilities are rare and they require some special skills to recognize unlike bugs. They studied chromium project's four additional releases, commit logs, code reviews and vulnerability entries. They found that files are less likely to be vulnerable if they have already been reviewed by developers who has been involved in vulnerability fixing. But also their results indicate that we cannot use the same reviewers for bug and vulnerability both due to their inexperience and non-familiarity of risk factors related to security (Meneely et al., 2014).

Another Study by Tantithamthavorn studied the reliability of the prediction model and stated that the quality of training data is very important in creating a good prediction model. Mislabeling of data can seriously compromise the quality of prediction model. They studied mislabeling in random and their impacts on the performance and interpretation of the defect predictive models. They have used Chromium Project as a data set to study the impact of mislabeling (Tantithamthavorn, McIntosh, Hassan, Ihara, & Matsumoto, 2015).

2.3 Bug-Vulnerability Relationship

Not a lot of research has been carried on this topic, we have found three papers that addressed this topic. Fonsecal and Vieira studied the software bugs that can lead security vulnerabilities, they studied 655 security fixed of 6 web applications. According to their results, generic software faults are responsible for all the security problems relating XSS and SQL Injections. They also found that another fault Missing function Call Fault Types is causing more than 76% faults relating to security. They analysis show that vulnerabilities relating to web applications are caused by software bugs which affect the restricted collection of statements. They have presented a detailed analysis of the code of fault models that cause

vulnerabilities in web applications (Fonseca & Vieira, 2008).

Camilo and Meneelyet studied the relationship between bugs and vulnerabilities. Bugs occur due to insufficient functionality whereas vulnerabilities are the abuse of functionality. They performed an in depth analysis on chromium project over 5 releases of the software over the time span of 5 years. They checked whether vulnerabilities are the consequences of the bugs. They states that before a vulnerability is exploited, it is present in the system for two years, hence they have studied the relationship over the interval of time period of two years. They mined the vulnerability data from the national vulnerability database and Common Vulnerabilities and exposures database whereas, for bugs, they approached the chromium bug repository. They used the logistic regression analysis, ranking analysis, developer experience, bug type classification and ranking analysis to check whether files which were changed for bugs are same as were the files changed for vulnerabilities. They concluded from the obtained results that there exists some correlation but the relation is weak. They also mentioned that security bugs had stronger relation with vulnerabilities than non security ones (Camilo, Meneely, & Nagappan, 2015).

=Munaiah and Camilo extended the research of (Camilo et al., 2015), they investigated further on the relationship between pre-release bugs and post release vulnerabilities, this time they checked the relationship for specific bugs like stability related bugs. An in-depth analysis was performed on chromium over the time span of six years with 5 major releases. The bug data set was divided according to the categories to study every category in detail for its relationship with post release vulnerabilities other than this, data set was pretty much the same. They evaluated the relationship using non-parametric null hypothesis test. They again used regression analysis to find relationship for categories of bugs, so that every category could be handled and analyzed individually. But, even after studying the relationship in depth, they still concluded that the relationship was weak overall and gave comparatively better results on the security related bugs. To generalize their results, they also performed the same analysis on a different data set Apache httpd, results were same, no significant relationship was found (Munaiah et al., 2017a).

Chapter 3

Proposed Approach

Since it's difficult to create a non-vulnerable system, a lot of research is being conducted on how to fix the vulnerabilities and how to prevent these vulnerabilities from being exploited. As software engineers, we need to find the cause of the vulnerability or the circumstances in which that vulnerability occurred. We also need to explore the hidden patterns of the vulnerability that which can provide us details about vulnerability behaviors.

In this chapter, we are going to explain our proposed approach that we are going to use for studying the relationship between bugs and vulnerabilities. We are going to discuss all the techniques that are being used in out approach.

3.1 Bug and Vulnerability Repositories

Software Bugs are faults in the system which are caused due to some coding mistake or insufficient requirement fulfillment. With the growth in scale, large software projects are likely to get more software bugs. These bugs are stored in a database called bug repository. These repositories not only store the information about bugs but also update the bug information whenever a change takes place in response to that bug. These repositories are important to be present in a system because they let the software quality researchers to investigate the root cause of these bugs and they also enable the researchers to create methodologies and techniques to prevent these bugs from occurring (Banerjee et al., 2017; Bozorgi et al., 2010; Dallmeier & Zimmermann, 2007; Jimenez et al., 2009; Kanwal & Maqbool, 2012). Today, big software vendors like Google [1][2], Firefox [3], and eclipse[4] are maintaining their respective bug repositories.

[1] https://bugs.chromium.org/p/chromium/issues/list
[2] https://developers.google.com/issue-tracker/
[3] https://bugzilla.mozilla.org/
[4] https://bugs.eclipse.org/bugs/

Some of them allow their bug data to be available freely to researchers (Xuan, 2012).

Software Vulnerabilities are security consequence of a system which are caused when security policy of a system is violated. Common Vulnerability and Exposures (CVE) is a list of names that are assigned to publicly known vulnerabilities. Every vulnerability that is published and accepted is given a unique CVE ID which uniquely identifies them (Gopalakrishna, Spafford, Gopalakrishna, & Spafford, 2005). Software Vulnerabilities are also stored in repository database along with the detailed information relating them. A software may or may not have it's separate vulnerability repository. But there are some online repositories which keep track of every information of every vulnerability that occurs in any system. Some of these vulnerability databases that are available online are National Vulnerability Database (NVD)[5], Common Vulnerability and Exposures (CVE)[6] [7]and Redhat [8]. But the updated repository that enlists all the attributes of every vulnerability available in all the vulnerability database or in other words, it assembles all the vulnerability attributes available in all repositories on one platform is CVEDetails (Munaiah, Camilo, Wigham, Meneely, & Nagappan, 2017b).

3.1.1 Bug Attributes

Bug attributes are the features of bugs that are found in bug reports, these bug attributes define the bug. They also reveal the reason how the system was broken, or why that bug occurred in the system. Bug attributes are important because they are used by researchers to propose fault localization techniques that can help analyze and predict the modules to be analyzed for new reported bugs. A list of useful attributes that are used by researchers are explained in Table 3.1(Nagwani & Verma, 2011)(Sharma, Kumari, & Singh, 2013)

Table 3.1: Bug Attributes

Attribute	**Explanation**
Bug ID	A unique number assigned to every bug to identify that bug in a system.
Bug Priority	It defines the importance of a bug as compared to others. Bug Priority 0 is considered the highest whereas 5 considered the lowest.

[5]https://nvd. nist. gov
[6]https://cve. mitre. org
[7]https://www. cvedetails. com
[8]https://www. redhat. com

Attribute	Explanation
Severity	This indicates how severe the problem is. e. g. trivial, critical, etc
Resolution	The resolution field indicates what happened to this bug. e. g. FIXED
# Comments	Bugs have comments added to them by users. it represents Number of comments made to a bug report.
Dependencies	Does the bug depend on other bugs to be fixed first or does it block fixing another bug ? The number of dependencies are recorded here.
Date of Close	The date when the bug was marked closed for any further change.
Keywords	The keywords that can be used to tag or categorize a bug.
Version	This defines the version of software in which the bug was found.
CC List	People who are informed about any change made in the bug, these people are directly or indirectly involve in fixing the bug.
Platform and OS	The computing environment where the bug was found is indicated here.
Bug Summary	It provides the brief description of the bug.
Changed Files	The files that were changed while fixing the bug in the system.
Component	This defines the component in which bug occurred
Bug Status	This field tells about the status of the bug whether bug is new, unresolved, or fixed.
Bug Publish/Create Date	The date when bug was published
Bug Fix Date	The date when bug was fixed and the stats of the bug was marked “Fixed”

Attribute	Explanation
Owner	Defines who identified and reported the bug.
# Attachments	Attachments can be anything which was attached by the reporter while reporting bug or attached by developer fixing the bug, it can be an image or some document.
Bug Fix Time	It indicates the amount of time that was taken to fix the bug.

3.1.2 Vulnerability Attributes

Every vulnerability is defined by some attributes that tell details about it's nature and the impact it has on any system. Vulnerability attributes don't vary from repository to repository, there are general vulnerability attributes present in Vulnerability repositories like CVEdetails, NVD and redhat. Use of these attributes by researchers may vary, they use the specific attributes which would help them achieve their goal. As mentioned before, CVEDetails assemble all the attributes of vulnerabilities at one place, so we are going to define it's attributes[9] (Özkan, 2012). Explanation is given in Table 3.2

Table 3.2: Vulnerability Attributes

Attribute	Explanation
CVE ID	CVE ID uniquely identifies a vulnerability that is publicly known.
CVSS Score	The Common Vulnerability Scoring System (CVSS) reflects the severity of a vulnerability which is captured using principle characteristic of the vulnerability. Score lies between 0-10 , 0 being the lowest and 10 being the greatest.
CVE Type	It defines the type of vulnerability.
CVE Summary	It provides a brief description about the vulnerability.

[9] www. cvedetails. com

Changed Files	The files that were changed while fixing vulnerability.
Gained Access Level	Which level of access was attained by the attacker when this vulnerability was exploited i. e. User, Admin or None.
Access	It defines the access type that attacker achieved whether it was local, local network or remote
CVE Publish Date	The date when vulnerability was officially published, the vulnerability might have existed in the system before that.
CVE Update Date	The date when system was updated after fixing the vulnerability.
Authentication	This defines whether attacker has to be authenticated to exploit this vulnerability or not.
Confidentiality	How much was the confidentiality of the system compromised? i. e. None, Partial or Full
Integration	Was the attacker able to modify the system using the specific vulnerability? If yes what is the level of modification i. e. None, Partial or Full.
Availability	Was the system availability compromised when this vulnerability was exploited?
Complexity	It defines the level of complexity a vulnerability possess *i. e.* Low, Medium or High

3.2 Association Rule Mining

Due to continuous increase in the size of data, we cannot analyze all the rules that occur in the system. To deal with this problem, an algorithm that could identify the associations between items in the form of rules was proposed in 1994 (Agarwal, Srikant, & Ahmad, 1994). This algorithm was used to discover the relationships between products in large transaction data in supermarkets. For Example The rule $\{$Item3, Item1$\} \Rightarrow \{$Item2$\}$ found in the sales data indicates that if a customer bought Item3 and Item1, he is likely to buy Item2 as well. Such information could be used for daily basis decision making like

Promotional Discounts, Items could be placed together so that if customer buys Item3 and Item1 he also sees Item2 nearby. Consider the data shown in Table 1, This is a small database of 5 transactions, The set of Items is {Item1, Item2, Item3} whereas 0 represents absence of an item and 1 represents presence of an item. {Item1, Item2}) {Item3} is an example of the rules in this small database, If A customer will buy Item1 and Item2, he is likely to buy Item3 as well.

Table 3.3: Transaction System with 5 Transactions and 3 Items

Transaction ID	Item1	Item2	Item3
1	1	1	0
2	0	0	1
3	0	0	0
4	1	1	1
5	0	0	0

{Item3, Item1} $\Rightarrow$ {*Item2*}

Association Rule Mining is a technique used to uncover the nature of relationship among different items. An association is a rule from Left Hand Side(LHS) to Right Hand Side(RHS). LHS and RHS consists of the unique, non-repetitive items. A set of items is known as item-set. The basic goal of association rule mining is to mine the associations between the set of items from a set of transactions (Zheng, Kohavi, & Mason, 2001). Let us now move to the formal definition of Association Rule Mining. Let I= $\{I_1, I_2, \ldots, I_n\}$ be the set if items, D be the set of transactions where I is the subset of T. An association rule is an implication of the form X->Y where X and Y are the item-sets (Agarwal et al., 1994).

Support

Support of a rule is defined as the support of the item set containing both the antecedent and consequent of a rule. The support of an item set is the probability of transaction in the database. An item set containing higher support is known as frequent item set. Support of Item3 is 2/5. Support of {Item1, Item2, Item3} is 1/5 (Zheng et al., 2001).

Confidence

Confidence is the minimum ratio of the support of the rule and support of the antecedent. The probability to find item or item set Y in a transaction, knowing item or item set X is in the transaction. Confidence

of $\{Item3\} \Rightarrow \{Item1\}$ = 1/2

Association Rule Mining was initially developed for market basket Analysis but right now it's being used in many other application areas including Web Usage Mining, Intrusion Detection and software engineering.

3.3 Proposed Approach for Finding relationships between Bugs and Vulnerability

The goal of the research is not to merely find relationship between bug and vulnerability but also to exploit that relationship for any interesting patterns that we can use to improve software security or software quality. While resolving a bug or vulnerability, all the information is stored into the bug or vulnerability repositories, this information consists of a series of attributes explained in Table 3.1and 3.2. According to the problem that researchers have to solve, they choose the attributes from repositories that will help them resolve the problem. Our first task is to create a link between the bug and vulnerability, we will select the attributes that will help us create a link. After establishing the link, we have to generate the association rules between the said entities. Second task of our research is to analyze the association rules for interesting patterns. In Figure 3.1, we have shown how our proposed approach is going to work.

The process that we are going to follow will be as follow

1. Attribute Selection
2. Changed/Commit Files Extraction
3. Converting Data into Basket Mode
4. Generating Rules using Association rule Mining
5. Post-Processing Association Rules

3.3.1 Attribute Selection

As discussed earlier in this chapter, we need to create a link between bug and vulnerabilities and for that we will use some attributes from bug and vulnerability repositories. One constraint in our research will be that we will only consider those attributes which will somehow be related in both repositories to create a link. Attributes are given in Table 3.4

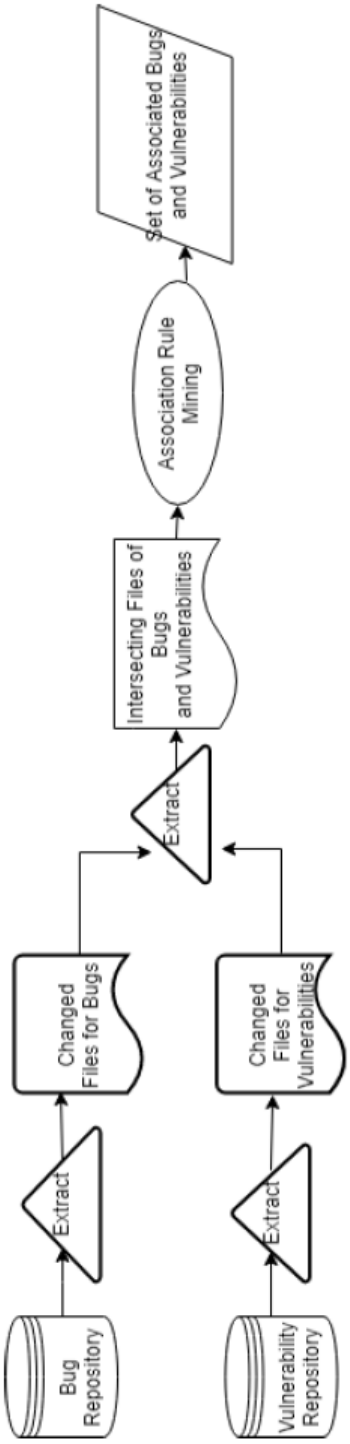


Figure 3.1: Proposed Approach for Finding relationships between Bugs and Vulnerability

Table 3.4: Attribute Selection for producing an associated list of bugs and vulnerabilities

	Attribute
Bug	Commit/Changed Files
	Bug Publish Date
Vulnerability	Commit/Changed Files
	Vulnerability Publish Date

3.3.2 Changed/ Commit Files Extraction

When a bug or vulnerability occur in a system, developers try to resolve that bug or vulnerability as soon as possible. While fixing the fault, some files of system are changed, these files that change in the process of bug or vulnerability resolution are known as changed or commit files. Commit Files are recorded in bug and vulnerability reports and are available in the fault repositories of the software.

Example of the bug changed file is given in Table *3.5* whereas sample of vulnerability changed files is given in Table 3.6

Table 3.5: Sample of Changed files extracted against every Bug

Bug ID	**Changed Files**		
Bug ID-001	file1	file2	
Bug ID-002	file3	file4	file5
Bug ID-003	file 3	file 4	
Bug ID-004	file 4	file 5	file 6

Table 3.5, represents the files that are recorded in the bug repository as changed files. These files were changed while resolving bugs given in Bug ID.

Table 3.6: Sample of Changed files extracted against every CVE ID

CVE ID	**Changed Files**		
CVE-ID-1	file1	file2	
CVE-ID-2	file2	file7	file8

In Table 3.6, files which were changed in order to resolve vulnerabilities given in CVE ID.

3.3.3 Merging the Files with Bug IDs and CVE IDs/Converting Data into Basket Mode

To create a link between bugs and vulnerability data, we need to convert bug and vulnerability data into a form which can further be used to extract association rules. We will use file names as the connecting variable, we will integrate all the changed files we have extracted from bug and vulnerability repository. We will place bug and vulnerability IDs against them to achieve the link. Now we will have files along with the bugs and vulnerabilities that occurred in them. A sample is given in Table 3.7

Table 3.7: Sample of Files with their corresponding bugs and vulnerabilities

File Name	Bug ID	Vulnerability ID
file1	Bug-ID-1	CVE-ID-1
file2	Bug-ID-1	CVE-ID-1,CVE-ID-2
file3	Bug-ID-2,Bug-ID-3	
file4	Bug-ID-2,Bug-ID-3,Bug-ID-4	CVE-ID-2
file5	Bug-ID-4, Bug-ID-2,Bug-ID-4	
file6	Bug-ID-4,Bug-ID-4	
file7		CVE-ID-2

Table 3.7 represents basket mode of the files, there are three types of commit files.

1. One which are fixed for bugs *e.g.* file 3 was only fixed for bugs Bug-ID-2 and Bug-ID-3
2. Second which are fixed for vulnerabilities only *e.g.* file 7 was just fixed for CVE-ID-2.
3. Third which are fixed for both bugs and vulnerabilities *e.g.* file 1 was fixed for Bug-ID-1 and CVE-ID-1

3.3.4 Association Rule Mining

Now that we have arranged data in the basket mode, we can extract rules from data. We will use Apriori algorithm for association rule mining. Here, we want rules from data set where bugs and vulnerabilities co occur in a file, along with co occurrence, we will also learn which bugs occur with certain vulnerabilities more than once. A rule has an antecedent and consequent in it, here we have bug ID as antecedent

and CVE ID as consequent, we will ignore files name in the rules to avoid ambiguity. The reason of using files was to get associated bugs and vulnerability, now that we have a list of bugs and vulnerability, we can ignore files.

A sample of data set is given in Table 3.8 showing how data set will look now.

Table 3.8: List of Bugs and Vulnerabilities

Bug ID	**CVE ID**
Bug-ID-1	CVE-ID-1
Bug-ID-1	CVE-ID-1,CVE-ID-2
Bug-ID-2,Bug-ID-3	
Bug-ID-2,Bug-ID-3,Bug-ID-4	CVE-ID-2
Bug-ID-4, Bug-ID-2,Bug-ID-4	
Bug-ID-4,Bug-ID-4	
	CVE-ID-2

3.3.5 List of Associated Bugs Vulnerabilities

A sample of Associated bugs and vulnerabilities is given below in Table 3.9

Table 3.9: Sample Association Rules

Rules
{Bug-ID-1==>{CVE-ID-1
{Bug-ID-1=>{CVE-ID-2
{Bug-ID-2=>{CVE-ID-2
{Bug-ID-3=>{CVE-ID-2
{Bug-ID-4=>{CVE-ID-2

3.3.6 Scoring

After we have extracted all the rules from data-set, rules are given a score that indicate their importance. Every rule is given a confidence value, a support value and the count(occurrence) of that rule in the data set. We will discuss a brief detail about these scoring. A sample Table of association rules with scores is given below in Table 3.10

Table 3.10: Scoring of Association Rules

Rules	Support	Confidence	Count
{Bug-ID-1==>{CVE-ID-1	0. 5	0. 3	10
{Bug-ID-1=>{CVE-ID-2	0. 3	0. 7	06
{Bug-ID-2=>{CVE-ID-2	0.4	1	08
{Bug-ID-3=>{CVE-ID-2	0.2	0.5	04

Support

Support exhibits co occurrence of a bug and vulnerability in the complete data-set. A high support may indicate that items have relationship between them while low value of support indicates that co occurrence can be by chance.

Confidence

The Confidence exhibits the co occurrence of bug and vulnerability that occur with each other frequently. A high confidence shows bug and vulnerability occur with each other all the time, whereas a low confidence might indicate that bug and vulnerability occurred with each other in rare case.

Count

Count displays the number of times a rule had appeared, how many times the same bug has appeared with the same vulnerability.

3.3.7 Post-Processing the Association Rules

We have another attribute that we selected to use to find association rules. One of the goal of our research was not only to exploit relationship of bug with vulnerability, we wanted to study two-sided relationship. We wanted to see if a vulnerability can cause any bug. The association rules we extracted are all showing relationship between bug to vulnerability. Here, we are going to consider a fact that if the bug has occurred before vulnerability, vulnerability is not the cause of bug, it can not be. Hence, to find which might have caused which, publish dates are very important. We extracted publish dates of all the bugs and vulnerabilities in all the rules, and then re arranged them in a way that if bug has occurred before vulnerability, rule will be bug⇒vulnerability else vulnerability⇒bug.

We extracted the dates from the bug and vulnerability reports. a sample of the rules after considering dates is given below in Table 3.11

Table 3.11: Sample of Final Bug and Vulnerability Rules

Rules	Support	Confidence	Count
{Bug-ID-1==>{CVE-ID-1	0. 5	0. 3	10
{CVE-ID-2=>{Bug-ID-1	0. 3	0. 7	06
{CVE-ID-2=>{Bug-ID-2	0.4	1	08
{Bug-ID-3=>{CVE-ID-2	0.2	0.5	04

3.4 Analyzing the Association Rules for Patterns

After extracting rules, we have selected a set of attributes that we are going to use to analyze and exploit the relationship between bugs and vulnerabilities, which are given in the Table3.12

Table 3.12: Attribute Selection for analyzing the associated list of bugs and vulnerabilities

	Attribute
Bug	Bug Type
	Bug Priority
	Bug Summary
Vulnerability	Vulnerability Type
	CVSS Score
	Vulnerability Summary

Figure 3.2 exhibits the attributes we selected for analyzing and the purpose we are going to use them for.

Using these attributes, we will analyze our entries of associated lists. We will compare the attributes of bugs and vulnerabilities to learn the hidden patterns. We chose these attributes because they are present in both the repositories, to compare attributes in two repositories, we must have comparable attributes in the repository. For Example we have first attribute Bug Type and Vulnerability Type, by comparing these attributes, we can learn which bug types occur mostly with which vulnerability types.

3.4.1 Bug Type and Vulnerability Type

We have our association rules now, to fine grain these rules and to get insights of the data, we need to analyze the attributes. The first attributes that we are going to analyze are Bug type and Vulnerability Type. Goal of analyzing these two attributes is that it might be the case that a specific type of bug

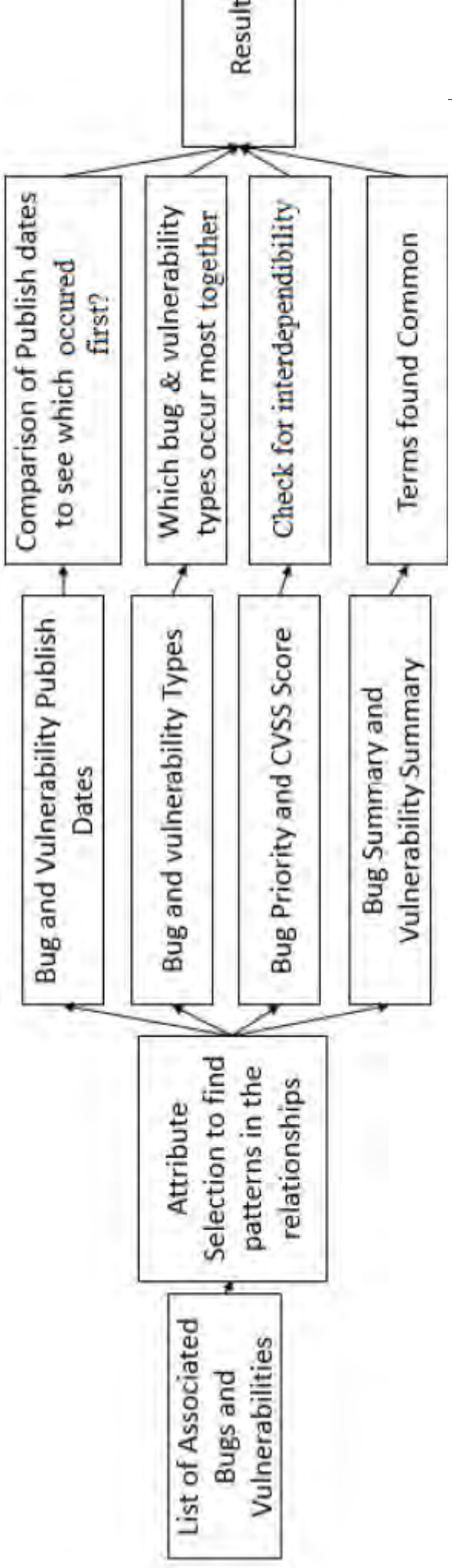


Figure 3.2: Analyzing Rule for Patterns

always resulted in a specific type of vulnerability e. g. One may assume that security bugs will always translate into DOS type of vulnerabilities. We also want to check which type of bugs have the most vulnerabilities and also which type of vulnerabilities have which type of bugs in parallel. A sample of the Bug and Vulnerability Types are given in the Table 3.13

We analyzed the attributes by making a inter-dependability Bug and Vulnerability type matrix to analyze which bug type occurs most with which vulnerability

Table 3.13: Sample of Bug and Vulnerability Types

Bug id	Bug Type	CVE ID	CVE Type
Bug-ID-1	Bug-Type1	CVE-ID-1	CVE-Type1
Bug-ID-2	Bug-Type2	CVE-ID-2	CVE-Type2

3.4.2 Inter-Dependability of Bug Priority and CVSS Score

Second attribute that we will study is Bug Priority and CVSS Score, goal of choosing these is finding out that if a bug and vulnerability is associated and the bug has high priority, will the vulnerability also be of high CVSS Score? We have made a Bug Priority and CVSS Score Inter dependability matrix to check which Bug priority is associated with which CVSS score more.

A sample of data is given in Table 3.14

Table 3.14: Sample of Bug and Vulnerability Scores

Bug id	Bug Priority	CVE ID	CVSS Score
Bug-ID-1	1	CVE-ID-1	5. 4
Bug-ID-2	3	CVE-ID-2	9. 3

3.4.3 Relationship between Bug and CVE Summary

Summaries provide a one liner description about the bug and vulnerability they are usually analyzed to find frequent terms used. We will analyze summaries for similar terms that are present in both bugs and vulnerability summaries given below in Table 3.15

Table 3.15: Summary of Bug and Vulnerability Summaries

Bug id	CVE ID	Bug Summary	CVE Summary
004	CVE-2008-003	abc. .	xyx. .

007	CVE-2016-005	ghj. .	rty. .

3.5 Summary

We have proposed an approach using which we can exploit the relationship between bugs and vulnerability. We will Association rule mining to find the rules of associated bugs and vulnerabilities. We will apply different techniques onto these rules to find any kind of pattern that could provide us some useful pattern. We will generate attribute comparison matrices to learn insights of the relationship.

Chapter 4

Experimental Setup

In this chapter, we will explain how we have setup the environment to run experiments using our proposed approach. We will discuss subject system, association rule mining approach and the analysis of the rules extracted from data set.

4.1 Subject System

We perform our experiments on google chromium project[1], we had two different repositories that we used for these experiments. For bugs, we used google chromium's own bug repository known as bug. chromium whereas for vulnerabilities, we used online vulnerability repository known as CVEDetails. Bug. chromium[2] is a repository that records all the issues that occur in the chromium project, these issues can be open, closed, new issues or issues to verify. We have taken all the bugs that have occurred in google chromium project from 2008 to 2018. In the same way, we also studied all the vulnerabilities that have occurred from 2008 to 2018 from CVEDetails [3]. Table 4.1 exhibits the details about the subject system.

Table 4.1: Subject System

Subject System	Number of Bugs	Number of Vulnerabilities
Google Chrome	4817	1576

[1]https://github. com/chromium/chromium
[2]www. bugs. chromium. com
[3]www. cvedetails. com

4.2 Data Collection

We collected data from two different repositories, One for bugs and one for vulnerabilities. Extracting data from both these repositories was different due to their structure so we are going to discuss both separately.

4.2.1 Bugs. Chromium (Bug Repository)

We had to extract two types of data from bugs. chromium. First, we had to crawl data of bugs along with its attributes. Secondly, we need commit files which were located inside every bug.

Bug Attributes Data Crawling

To crawl general bug data with its attributes, we had to go to the website and the website offered to download all the data in an ordered and systematic form. We selected the attributes we needed for our research methodology. Data was provided in CSV format.

Bug Commit Files Data Crawling

For commit data, we designed a crawler in python, the crawler would open every bug ID. After opening the bug ID in bugs. chromium, it searched for commit and opened the link straight after commit. It crawled all the commit files from the page after opening the commit link. To write our crawler we used ”scrapy” framework written in python [4]. It is an open source framework for extracting data from websites. By default it handles 16 concurrent requests. Instead of doing it all in single bunch of code, we split our problem in two modules that can work independently. At any stage of the project, change in one module will not effect the functionality of other module. First module reads Bug IDs of chromium project from a CSV file. If that Bug ID contains any commit, the module stores its ID and URL into another file named commitbugs. csv. Last module reads commit IDs from commit. csv and using dynamic URLs iterate through each of the commit. From each commit, it gets the listed file and store in our final version of document ”crawlbugs. csv”.

4.2.2 CVEDetails (Vulnerability Repository)

We had to extract two types of data from CVEDetails as well. We had to crawl all the data of vulnerability with all its attributes. Secondly, we needed commit files that were changed while fixing

[4] https://scrapy.org/

vulnerabilities.

Vulnerability Attributes Data Crawling

For Attribute data, we opened the Chromium product data from the website and downloaded it.

Vulnerability Commit Files Data Crawling

Due to confidentiality and privacy concerns, commit files data was not provided in any vulnerability repository. We have discussed before, bugs. chromium didn't only contain data of bugs but all the issues that chromium project encounters over the time. We searched for all the vulnerability IDs in the repository and then looked for the commits and crawled commit files.

To write vulnerability data crawler, we used "scrapy" framework written in python as well [5]. In the first module of this crawler, it reads CVE IDs of chromium project from a CSV. Using dynamic URL, it access the content of issue list filtered with given CVE ID. Our crawler then reads all the bug IDs and URLs from that page and stores them into another file name bugs. csv. Second module reads the bug IDs from bugs. csv. Using dynamic URLs get the content of individual bug. If that bug contains any commit, the module stores its ID and URL along with CVE ID into another file commitvulnerability. csv. Last module reads commit IDs from commit. csv and using dynamic URLs iterate through each of the commit. From each commit get the listed file and store in our final version of document "crawlvuln. csv".

4.2.3 Converting Data into Basket Mode

After extracting commit files for both bugs and vulnerabilities, we had two files *i. e*. bug IDs along with commit files, vulnerability IDs along with commit files. To merge bugs and vulnerability files, we used commit files as base attribute. We extracted all the commit files into another CSV and then added the corresponding BUG ID or CVE ID. As a result, some files were fixed for bugs while some were fixed for vulnerabilities. Some files were also fixed for both. These files are known as intersecting file between bugs and vulnerabilities and represented co-occurrence of both bugs and vulnerabilities.

4.2.4 Applying Association Rule Mining

Association rule mining is used to find the frequent rules from data set. In our case, we wanted to extract rules from data set which contain both bugs and vulnerabilities. In the last step, we converted out data

[5] https://scrapy.org/

into basket mode and saved the data into a CSV file. We removed file names from the file, because we just wanted to create associations between bugs and vulnerabilities, File names are no use for us now. We applied Association Rule Mining in R Studio [6]and the commands we used to find rules are given as follow

install. packages("arules");

Arules is a package in RStudio which provides the infrastructure for representing, manipulating and analyzing transaction data and patterns. It is used to find frequent item sets and association rules from data. We will use "Arules" package to find association rules from our data set [7]

library(arules)

After installing the package, we are going to to add the library to use the functions from this library. This command is used for adding Arules.

trans = read. transactions("T:/IncreasedBugDataDataset/dataforrules. CSV", format = "basket", sep=", ");

Now that we have loaded arules, we can use its functions, we are going to load out data set from which we need to find association rules. The data set is imported using this command.

inspect(trans)

This command is used to check the dataset we have loaded.

rules <- apriori(trans, parameter = list(minlen=2, maxlen=2, maxtime=5, confidence=0. 00001, support=0. 00001))

This command is used to find association rules from dataset. Apriori algorithm is used here, parameters are defined to get specific rules from the data. Rules we are going to extract should have exact length of 2, confidence and support should be 0. 00001, the lesser the confidence and support will be the maximum rules it will extract. The confidence and Support Value was set to the minimum value to make sure that all the rules that exist in the system are unveiled.

[6]https://www.rstudio.com/
[7]https://cran. r-project. org/web/packages/arules/index. html

rules_subset <- subset(rules, (lhs %pin% "Bug ID="));

This command will find the subset from rules which have LHS as "Bug ID" and will ignore all the rules which have CVE ID as LHS. We are doing this to remove duplicates, at this stage bug1->CVE1 IS SAME AS cve1-> bug1.

inspect(rules_subset)

We will check our rules by using this command.

write(rules_subset, file ="myRulesBUG. csv", sep = ", " , quote= TRUE, row. names = FALSE)

We will write our rules using this command to a CSV file. Write is a method used to write output data to a file, which will give the name of CSV File to store the rules and their attributes.

In above command, write is a method name which takes the rules as argument, name of csv file in which we are needed to store the calculated rules, and some other specifications.

4.2.5 Post Processing Association Rules

Since, we are studying two sided relationships and analyzing which entity might foreshadow the other one. We are not stating causal relationship but we are re-arranging our rules according to publish date to check which entity might foreshadow the other one. To study this, we required the publish dates of bug and vulnerability. We assumed that if bug publish date is less than vulnerability publish date, it means bug might be triggering vulnerability to occur else vulnerability might be causing the bug. To do this, we concatenated bug and vulnerability dates with bug IDs and CVE IDs and using a java program we changed the the positions of bug and CVE IDs. Example is given in the Table 4.2

Table 4.2: Post Processed Association Rules

Current Rule	**Rule concatenated with Bug and Vulnerability Publish Date**	**Post Processed Rule**
{Bug ID=003} => {CVE ID=CVE-2016-001}	{Bug ID=691099@#Bug Date=18-May-2017} => {CVE ID=CVE-2016-5218@#CVE Date=19-Jan-2017}	{CVE ID=CVE-2016-001=>{Bug ID=003

{Bug ID=004} => {CVE ID=CVE-2017-002}	{Bug ID=767761@#Bug Date=22-Sep-2017} => {CVE ID=CVE-2017-5086@#CVE Date=27-Oct-2017}	{Bug ID=004} => {CVE ID=CVE-2017-002}

4.2.6 Extracting Selected Attributes Data for Bugs and Vulnerabilities in Rules

After post processing association rules, we had to analyze the patterns in those rules. To do that, we needed the attributes of bugs and vulnerability. we extracted those attributes using a java program from already crawled data of bugs and vulnerabilities along with their attributes in the first step.

4.2.7 Analyzing Selected Attributes for Patterns

4.2.7.1 Bug Type and CVE Type

To analyze Bug type and CVE type, we created a Bug/CVE Type matrix to study inter dependability of Bug Type and CVE Type. We studied which bug type occurs most with which kind of vulnerability using the matrix.

4.2.7.2 Bug Priority and CVSS Score

We studied this relationship by making Bug Priority and CVSS Score matrix and studied which bug priority bugs contain more vulnerabilities and which vulnerabilities are more likely to appear with which priority of the bugs. This could provide us insight of whether Bug Priority and CVSS Score are generally related and also if not then are there some patterns that can be of interest for us.

4.2.7.3 Bug Summary and CVE Summary

We analyzed summaries for the common words in the associated bug and vulnerability. For that, we compared both the summaries and the terms that were similar in both the summaries we highlighted them and later extracted the highlighted terms out in another column.

4.3 Summary

We have stated the setup for all the experiments we will perform to explore relationship between bugs and vulnerabilities. We not only have exploited the relationship but we have also analyzed the relationship by the attributes as well to study all the possible aspects of the relationship as well.

Chapter 5

Experimental Results

We have conducted an in-depth exploratory analysis of the relationship between software bugs and software vulnerabilities of Google Chromium project and we have identified some patterns that can help to improve software security and software quality. In this section, we are going to present those results and discuss the findings of the research

5.1 Results and Discussions

5.1.1 Research Question 1: Are Files fixed for Bugs or Vulnerabilities likely to be fixed for future Bugs or Vulnerabilities?

Motivation:

We are studying Google Chromium project for exploring relationships between bugs and vulnerabilities. This research question serves to measure the relationship between software bugs and software vulnerabilities in general. Software Security is one of the major concerns of software vendors these days. A lot of research is conducted for identifying and fixing the cause of software security breaches. Software Vulnerabilities and software bugs are both defects of software, as we have discussed earlier. At many points of time, software vendors found cause behind a major vulnerability to be a small coding error or requirement bug which went uncaught by software development team. We analyzed whether in general also, there is some relationship between software bugs and software vulnerabilities. We also analyze whether just a bug can be the cause of vulnerability or can a vulnerability also cause bugs to occur?

Analysis:

We used Association Rule Mining to investigate if some significant association exists between software bugs and software vulnerabilities in Google Chromium Project. There were total 3, 862 bug and vulnerability co-occurrence rules found in chromium project. Out of 3, 862 rules, 2998 rules stated vulnerabilities occurred before bugs while in remaining 866 rules bugs occurred first. A sample of results is given in Table 5.1

Table 5.1: Sample of Discovered Bug and Vulnerability Rules

Rules	Support	Confidence	Count
{CVE ID=CVE-2017-5086=>{Bug ID=784761	0. 001361934	0. 25	12
{Bug ID=770946=>{CVE ID=CVE-2017-15386	0. 001248439	0. 169230769	11
{CVE ID= CVE-2017-0663=>{Bug ID=801488	0. 001134945	0. 12987013	10
{Bug ID=706053=>{CVE ID=CVE-2017-5110	0. 000907956	0. 307692308	8
{CVE ID= CVE-2017-5061=>{Bug ID=815187	0. 000794461	0. 189189189	7

We have used 3 types of measures in these rules, we have Confidence, Support and Count. We have set our Confidence and Support to the least value to obtain all the co-occurrences of software bugs and Vulnerabilities in the system. For our data-set, Support values for all the rules are generally found low. Highest support was found to be 0. 0040 for a rule that occurred 36 times in the system. If a bug and vulnerability have occurred together for 36 times in a 10 year time span, it means relationship is significant although . 004 support value seems low. On the other hand, the maximum confidence value is 1, high confidence values denote that the entities are most likely to be found with each other only.

In google chromium project, {CVE ID= CVE-2017-15412=>{Bug ID=801488 rule occurred 36 times and has the highest support 0. 0040, CVE ID 2017-15412 is a reserved vulnerability, it has not been defined on the vulnerability database due to being reserved by an organization or person [1]whereas the bug 801488 [2] is a regression bug which occurs when text on bookmark page is seen chopped after resizing browser window. The confidence value for this rule is 0. 46 means bug and vulnerability in this rule also occur with other bugs and vulnerabilities as well. We can analyze the cause of this rule for its multiple occurrence together if we know the description of the vulnerability as well, which is confidential. The reason for making this vulnerability and its information reserved can be that it was causing so many bugs to occur. Sample of high support values are given in Table 5.2

[1]https://cve. mitre. org/cgi-bin/cvename. cgi?name=CVE-2017-15412

[2]https://bugs. chromium. org/p/chromium/issues/detail?id=801488&can=2&q=801488&colspec=ID%20Pri%20M%20Stars%20ReleaseBlock

Table 5.2: Rules with High Support

Rules	Support	Confidence
{CVE ID= CVE-2017-15412=>{Bug ID=801488	0. 004085802	0. 467532468
{CVE ID=CVE-2015-6780=>{Bug ID=675239	0. 00226989	0. 666666667
{CVE ID= CVE-2017-15420=>{Bug ID=777419	0. 001702417	0. 555555556
{CVE ID=CVE-2015-6780=>{Bug ID=700591	0. 001588923	1
{CVE ID= CVE-2017-15951=>{Bug ID=784761	0. 001361934	0. 25

Second and third rule in the Table 5.2 have relatively high support with high confidence, which means the count of the rule is high as well as that the bug and vulnerabilities in this rule has also occurred with some other bugs and vulnerabilities. Whereas count of 4th rule is high as well as the confidence, which means bug and vulnerability in this rule has only occurred with each other multiple times. The vulnerability CVE-2015-6780 is called User After Free Vulnerability, this vulnerability caused DOS attacks and Bug ID 582720 can be the bug that attacker added in the system.

For 171 rules, highest Confidence value is 1 5.3sample is given in Table

Table 5.3: Rules with High Confidence

Rules	Confidence	Support
{CVE ID=CVE-2015-6780=>{Bug ID=700591	1	0. 001588923
{Bug ID=477970=>{CVE ID=CVE-2015-1274	1	0. 000113494
{CVE ID=CVE-2017-5078=>{Bug ID=809062	1	0. 000113494
{CVE ID=CVE-2015-1260=>{Bug ID=583720	1	0. 000113494
{CVE ID=CVE-2014-7936=>{Bug ID=475971	1	0. 000113494

In rule 1 in Table 5.3, CVE-2015-6780 has occurred 14 times with Bug ID 700591 and they have occurred only with each other which makes it a very significant rule. The vulnerability CVE 2015-6780 is called User-after-free vulnerability. This vulnerability allowed remote attackers to cause DOS attack whereas bug 700591 triggers some “git cl format” discrepancies which is a security bug. It is possible that this vulnerability could have created discrepancies and some bugs and Bug 700591 is one of those bugs. Other rules in the Table have high confidence value and low support value, in these rules the bug and vulnerability has only occurred together once, which can be co-incidental.

In Figure 5.1, we have provided some important distributions of our research. In the chromium

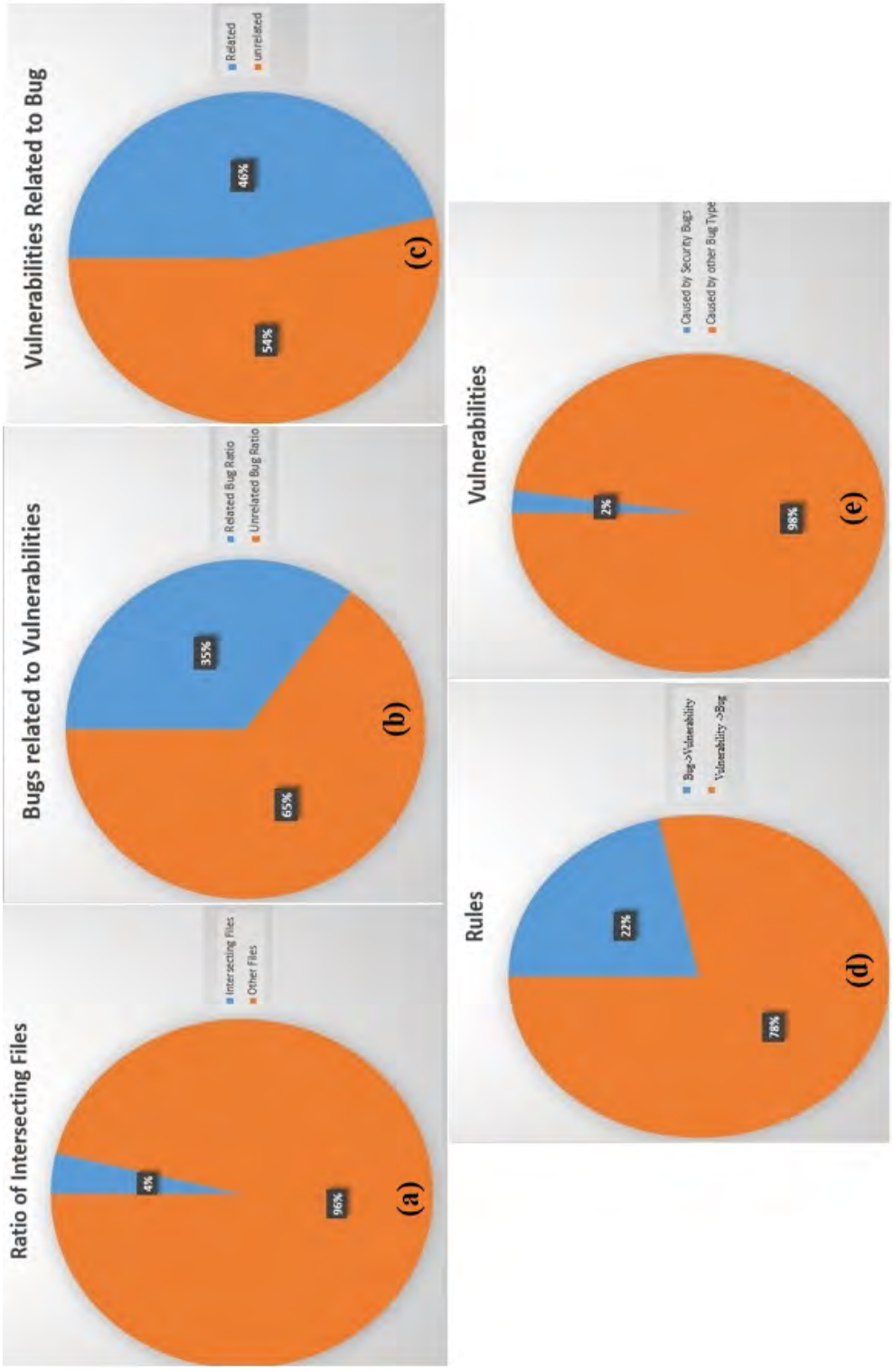


Figure 5.1: Distributions of Results

project, some bugs and vulnerabilities didn't have commit files due to security issues, total 4817 bugs were reported out of which 1687 bugs had commit files available. In the same way, out of 1576 vulnerabilities commit files for 397 vulnerabilities were available in the google chrome repository. We placed the files against bug and CVE IDs to find the files which were responsible for both the bug and vulnerability occurrence. Figure (a) represents 269 intersecting files which contain both bugs and vulnerabilities in them. In (b), 46% of the vulnerabilities were found co occurring with bugs whereas in (c), 35% of the bugs were co occurring with vulnerabilities. Out of 3, 862 rules, 2998 rules stated vulnerabilities occurred before bugs while in remaining 866 rules bugs occurred first depicted in (d) and (e). Hence, we can say that vulnerability ->Bug association is strong. More Vulnerabilities foreshadow future bugs for currently available file in the Chromium Project.

5.1.2 Research Question 2: If a file has one vulnerability, then how likely is it to have another vulnerability in it? Do files with more bugs have higher vulnerability?

Motivation

In previous research (Neuhaus et al., 2007), it was stated that in Mozilla system when a component had one vulnerability it was unlikely for that component to have any more vulnerabilities. We wanted to check it is really unlikely for a file to have future vulnerability if it is already fixed for some previous vulnerability. Second part of this research question aims to study how a good bug prediction can help at finding vulnerabilities. We wanted to study whether "bugginess" of files could be used as indicator of vulnerable files in a system as well.

Analysis

In the case of Google Chromium, there were many files which were fixed for more than one vulnerability. There were total 561 commit files that were changed while fixing some vulnerabilities. Out of these files, 82 files had more than one vulnerability in them, some of the files were fixed for around hundred of vulnerabilities in google chrome. Files that were fixed for 10 or more vulnerabilities are exhibited in Table 5.4

Table 5.4: File Rank List for Vulnerabilities

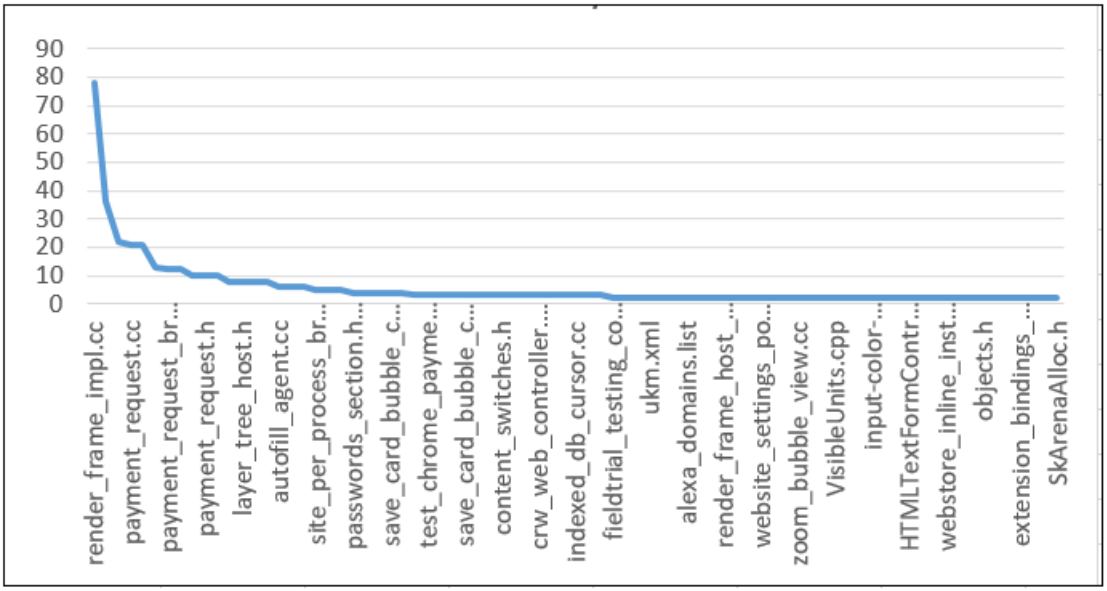
Files	Vulnerabilities Count
render_frame_impl. cc	78

manifest. json	36
enums. xml	22
payment_request. cc	21
navigator_impl. cc	21
md_settings_localized_strings_provider. cc	13
payment_request_browsertest_base. cc	12
payment_request_browsertest_base. h	12
PaymentRequestImpl. java	10
payment_request. h	10

The files given in the above Table for bugs and vulnerabilities are changed multiple times, we analyze these files for changes made in response to software bugs and vulnerabilities.

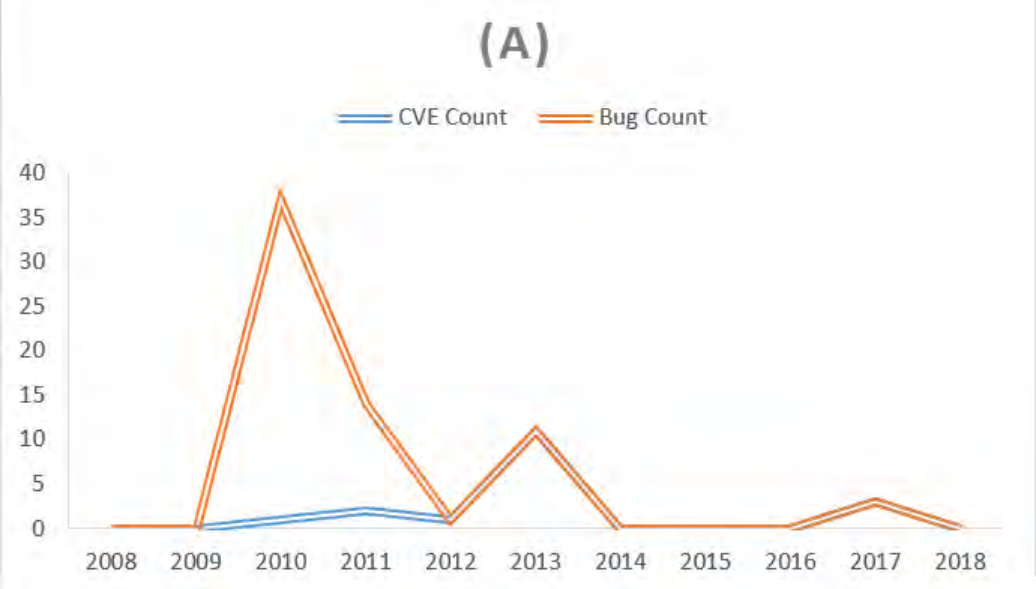
In Figure 5.2, we have displayed commit files along with the vulnerability count in them. According to a research, files that have one vulnerability are not likely to have another in it. We have file "render_frame_impl.cc" that was fixed for 78 different vulnerabilities whereas minimum number of vulnerabilities found in a file are 1. Out of 559 vulnerability commit files, Only 82 commit files were fixed for more than one vulnerabilities whereas rest 480 of them were fixed only once for one vulnerability. These files can be used to create a good vulnerability prediction model. Maintainers should focus more over the commit files in which vulnerabilities are occurring again and again to reach the cause of vulnerabilities.

Figure 5.2: Overview of Vulnerability Count in Files



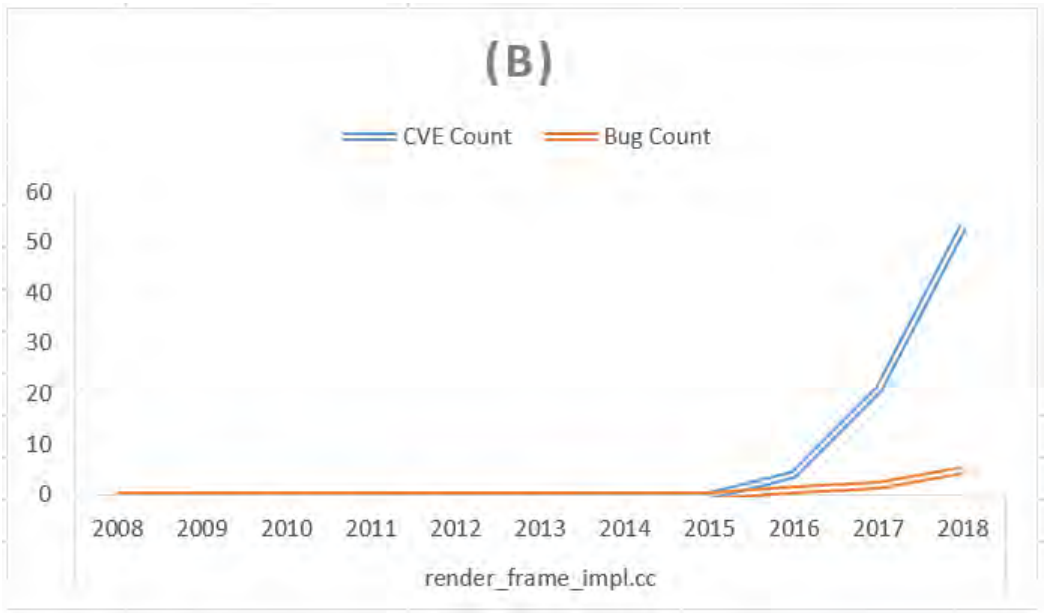
In previous research it was stated that a bug takes 2 years to be translated into a vulnerability (Munaiah et al., 2017b). We wanted to analyze if that hypothesis is true. Due to our inability to present all the files here, we are going to represent four files which have both bugs and vulnerabilities in them. We are analyzing file which has high number of bugs, one which has high number of vulnerabilities other files are having moderate level of bugs and vulnerabilities.

Figure 5.3: Timeline Analysis for Bugs and Vulnerabilities File 1



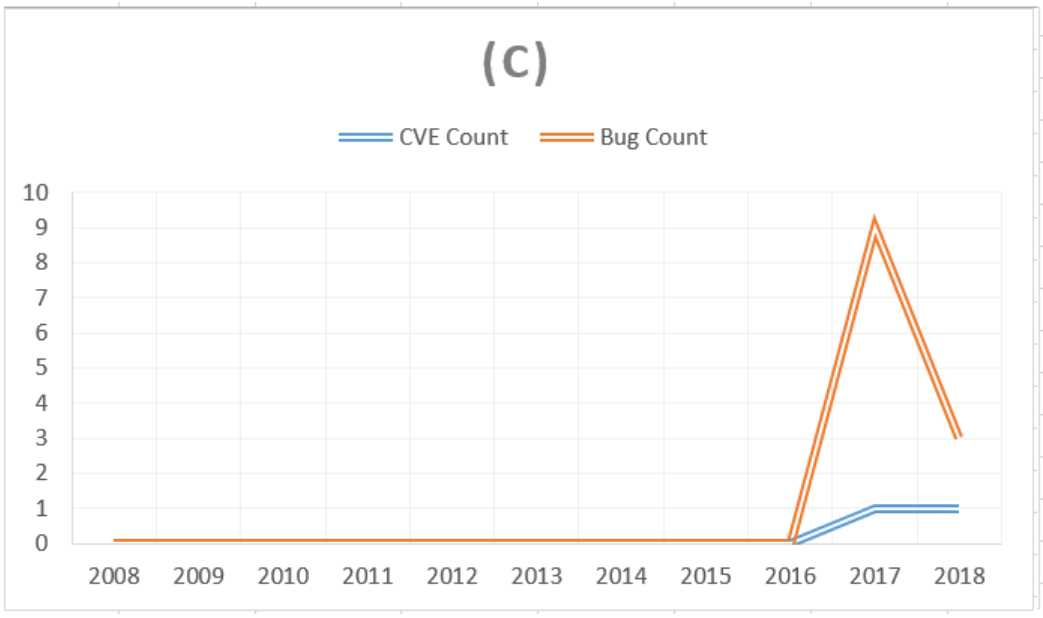
In file enum. xml, most bugs occurred in 2010 while most vulnerabilities have occurred in 2013. When count of bugs is high, count of vulnerabilities is low, whereas when count of vulnerabilities is high, count of bugs is low. The file was fixed multiple times while resolving bug. If we analyze the file on the span of two years, 2008 and 2009 had no bug there was 1 vulnerability in 2010 and 2 in 2011. 2010 had 36 bugs and 1 vulnerability in 2012. 2011 had 12 bugs and 0 vulnerability in 2012. and then there were vulnerabilities found in 2017. Hence, we can state that vulnerabilities are not foreshadowed by bugs after 2 years. With the world going so fast and the hackers technologies going even faster, they won't leave anything exploitable to get access to any system.

Figure 5.4: Timeline Analysis for Bugs and Vulnerabilities File 2



File render_frame_impl. cc has maximum vulnerabilities in 2018, the file has had maximum vulnerabilities from 2016-2018 and requires a lot of attention. First vulnerabilities were found in 2016 and there was one bug found that year as well. In 2016, 53 vulnerabilities were found. The reason of increase in the amount of vulnerabilities might be the vulnerabilities that occurred in 2016 or the bug that occurred in 2016 as well. While studying this file for fix, maintainers of the system should study the bugs and vulnerabilities that occurred in 2016 as well to fix the file for good.

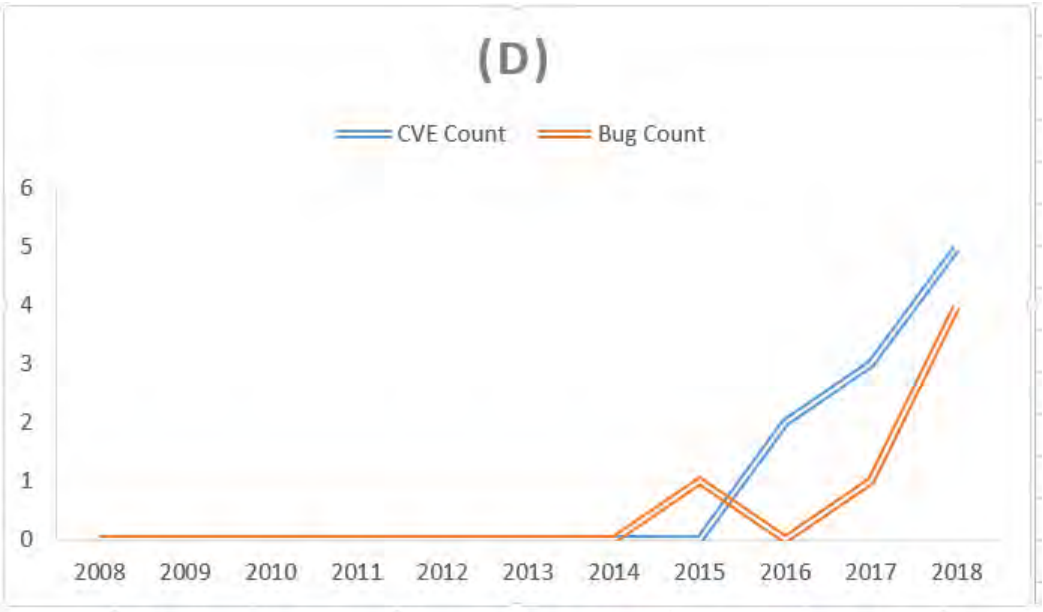
Figure 5.5: Timeline Analysis for Bugs and Vulnerabilities File 3



The third file payment_request. h is also displaying number of bugs and vulnerabilities are increasing

here as well. Although this file is less vulnerable than others, vulnerabilities were found first in 2017 when the file also had 9 bugs in it. While in 2018, although number of vulnerabilities decreased but at the same time number of bugs also increased.

Figure 5.6: Timeline Analysis for Bugs and Vulnerabilities File 4



First vulnerabilities occurred in 2016, in 2014 no bug was fixed in this file whereas in 2017, 3 vulnerabilities were found which could have been caused by the bug in 2016. In 2018, amount of bugs and vulnerability increased. The number of bugs and vulnerabilities in this file are also increasing, this file needs to be handled for bugs and vulneabilities.

To summarize the results, one might notice from the graphs that google chrome project is getting more and more buggy and vulnerable with time. The system might have some major quality and security issues if some proper measures were not taken to cope with security and quality issues.

In second part of question, we have studied the files that were fixed most for both bugs and vulnerabilities. To do so, we checked top 20 files of vulnerability rank list, and checked if they were also fixed as the most buggy files for bugs as well. We achieved only 5% common files in bugs and vulnerability rank lists. Hence, our verdict is that some of the buggiest files can be vulnerable but same prediction models will not provide us optimal results. Rank lists for Bug and Vulnerability rank lists are given below in Table 5.5

Table 5.5: Sample of File Rank List for Bugs

Files	Count

histograms. xml	61
about_flags. cc	60
generated_resources. grd	54
enums. xml	48
flag_descriptions. cc	39

Table 5.6: Sample of File Rank List for Vulnerabilities

Files	Count
render_frame_impl. cc	78
manifest. json	36
enums. xml	22
payment_request. cc	21
navigator_impl. cc	21

5.1.3 Research Question 4:Do Security bugs only translate into the vulnerabilities? Are vulnerabilities more likely to be found in High Priority Bugs?

Motivation

Since Vulnerabilities are security consequences of bugs, one might assume that security bugs translate into vulnerabilities. So we wanted to study whether any other bug types other than security bugs can also translate into vulnerabilities or not.

Analysis

Out of 1576 vulnerabilities, only 36 of them co-occurred with security bugs. Most of the vulnerabilities co-occurred with the general type “Bug”. Which means bugs other than security are more often translated into vulnerabilities. The reason can be that when some security bug occurs in a system, it is fixed on priority due to its likelihood of being translated into vulnerabilities.

We created a Bug and Vulnerability Type matrix to find which bug and vulnerability type co-occur most. The bug type “Bug” contains most vulnerabilities of type “DOS +Overflow” and “Bypass. “Bug Regression” contains maximum DOS+Overflow vulnerabilities whereas “Bug Security” has the least vulnerabilities in it with maximum Bypass vulnerabilities. Again DOS +Overflow are found most in

Feature and Task bugs as well. The Bug and Vulnerability Type Matrix is given in Table 5.7

For second part of question, no strong connection was found between attributes Bug Priority and CVSS Score. Relationship is represented in Table 5.8

Table 5.8: Analysis of Bugs and Vulnerability Scores

Bug Priority/CVSS Score	**0-2. 5**	**2. 5-5**	**5. 0-7. 5**	**7. 5-10**
0	0(0%)	22 (48%)	23(52%)	0(0%)
1	0(0%)	472(41. 65%)	636(56. 13%)	25(2%)
2	0(0%)	566(42. 8%)	731(55. 3%)	23(1. 7%)
3	0(0%)	496(37. 63%)	802(60. 8%)	20(1. 5%)

All the bugs having “0” and “1” “2” and “3” priority have most associated CVSS Score 5. 0-7. 5.

5.1.4 Research Question 6: Are Associated Bugs increasing the count of Vulnerabilities in Files? And are associated vulnerabilities increasing the count of Bugs in Files?

Although in general, count of vulnerabilities is high in files. But when we check the vulnerability count in in the files where bugs don’t exist, the highest count of vulnerabilities in a file remains as low as “4” whereas the lowest count is 1. Commit files of source language C are the highest count files that were changed for vulnerabilities having no associated bugs. The figure 5.7 represents the files having only vulnerabilities with zero bugs. The count of vulnerabilities in files is low where bugs are not involved. Hence, one might assume that increase in the amount of vulnerabilities might be due to associated bugs.

In the same way, when we just checked count of bugs in files having no vulnerabilities, count didn’t change alot. The highest number of bugs in files were still 61 and apart from that too , the count was pretty much same. The highest number of bug file is an XML file but alot of C files are also present in this.

Table 5.7: Bug and Vulnerability Types Matrix

	Type 1[a]	**Type 2**[b]	**Type 3**[c]	**Type 4**[d]	**Type 5**[e]	**Type 6**[f]	**Type 7**[g]	**Type 8**[h]	**Type 9**[i]	**Type 10**[j]	**Type 11**[k]	**Type 12**[l]	**Type 13**[m]
Bug	303	2	275	1	17	17	54	377	59	114	4	124	19
Bug-Regression	60	0	49	1	7	7	21	124	40	9	1	13	3
Bug-Security	20	0	2	0	0	0	2	2	3	2	1	2	2
Feature	46	0	43	0	4	4	12	74	3	24	0	25	6
Task	16	0	16	0	2	2	6	33	0	9	0	10	1

[a] Type1=Bypass
[b] Type2=Directory Traversal
[c] Type3=DOS
[d] Type4=DOS Executive Code
[e] Type5=DoS Executive Code Overflow
[f] Type6=DOS+Info
[g] Type7=Memory Corruption
[h] Type 8=DOS +Overflow
[i] Type 9=Executive Code
[j] Type 10=Executive Code Overflow
[k] Type11=Executive Code XSS
[l] Type12=Overflow
[m] Type1=XSS

Figure 5.7: Count of Vulnerabilities with no Bugs

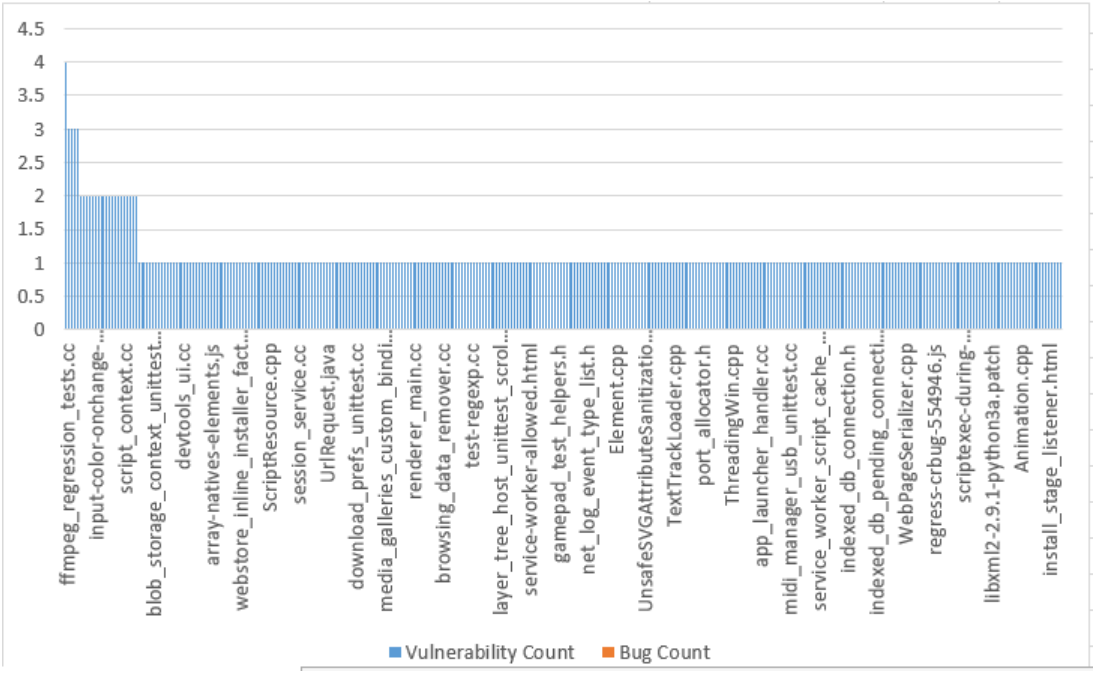
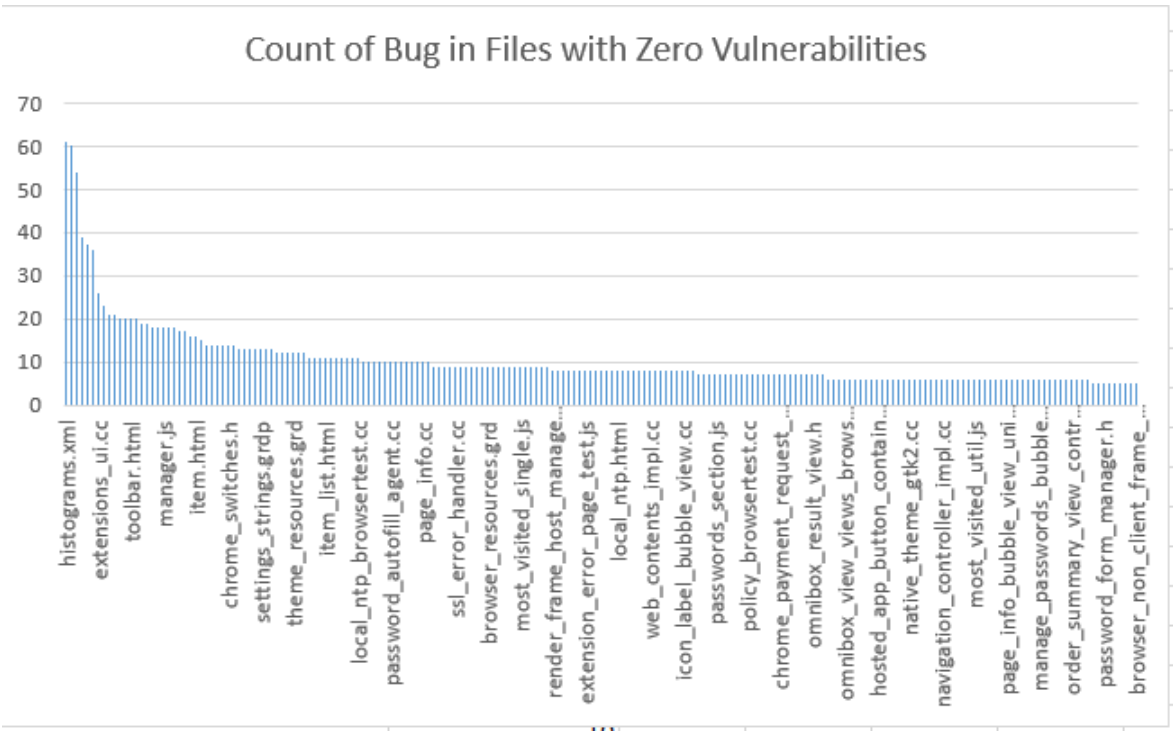


Figure 5.8: Count of Bugs with no Vulnerabilities



5.1.5 Patterns in Google Chromium Project

We studied google chromium project in detail to explore any relationship between two kind of defects, software bugs and software vulnerabilities. We studied if any bug could foreshadow any vulnerability

or any vulnerability can cause some bugs to occur. Google Chromium has been of great interest for researchers in recent times due to their "Chrome Reward Program" which gives rewards to researcher who invest their time for making Chromium Projects secure.

We not only studied the relationship between software bugs and software vulnerabilities but also we tried to find some patterns that could help chromium improve their security system. While studying system we found that C, C++ Files are most buggy files in chromium according to their history. These files were fixed again and again for bugs and vulnerabilities both, so these files should be tested adequately to prevent any kind of future security or quality failure. DOS and Overflow Vulnerabilities mostly occur with bugs. Most of the buggy and vulnerable files are from C and c++, because of the inability of C and C++ to check for overflow which can be one reason whereas java and python are other source code languages which do not have abundant bugs and vulnerabilities in them.

Buggy and Vulnerable Files have same behavior, if a file will have more bugs in past it will be called more suspicious in future. In the same way, if a file has more vulnerabilities it should be be called more vulnerable for future as well. But Same Bug Prediction Model can not be used to detect new vulnerabilities, a good bug prediction model can not judge a file for vulnerability.

Bugs do not require a time span to convert into vulnerability, before it was stated that for bug to be exploited to get some breach inside the system two year time span is required. No, with the advancement in the field of security and in the field of hacking, every second many attacks are generated to get some point of entry into the system.

In our research, we have explored every bug and vulnerability specifically, we have studied the possible relationship between bug and vulnerability. We have used all the available commit files for this. We have used association rule mining, that has not only given us the co-occurrence of software bugs and vulnerability but also provided us details how significant is the rule. We have also explored other attributes such as Bug and Vulnerability Types, Bug priority and CVSS Score and Bug and Vulnerability Summaries for patterns that could support our hypothesis.

We have found a significant amount of relationship between bugs and vulnerabilities, we have explored the relationship in both directions, from Bug to Vulnerability and from Vulnerability to Bug. 35% Bugs were found related to Vulnerabilities whereas 46% of the vulnerabilities were found related to bugs. We have also found some patterns that could be used in future research to improve software quality and software security

This research can help maintainer of Google Chrome while making a vulnerability Prediction System, he can use these patterns to make the system more secure. He can put more focus on C and C++

files. A we have discussed in the start, many vulnerabilities are reported to be caused by some bug in the commercial market, so we can make a vulnerability prediction model for the vulnerabilities that are specifically associated with some bugs. We can study if those bugs are causing the vulnerability and can help the bug information fix vulnerability as well. When a vulnerabilities is reported, it should be checked for associated bugs using the similarity matrices. If there exists any associated bug, the commit files of both the bugs and vulnerabilities should be fixed to prevent any future vulnerability.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this chapter, we are going to conclude our research, we are going to conclude our research and discuss how our results were benificial for Google Chromium in terms of software security and software quality.

Due to increased usage of software, software quality and software security have become two great concerns of software engineers. Efforts are made to improve software quality and software security, we have made another effort by studying the relationship between two major attributes that effects software quality and software security *i.e.* software bugs and software vulnerabilities. We have studied software bugs and software vulnerabilities for any relationship that occurs between them.

We have studied this relationship on Google Chromium Project. Our results show that there exists empirical association between software bugs and software vulnerabilities in Google Chromium but association is not very strong, 35% of bugs were related to vulnerabilities whereas 46% of the vulnerabilities were found to be related to bugs. Vulnerabilities->Bug relationship was more than Bug->Vulnerability.

We also found some interesting patterns that could help software engineers of Google Chromium Project to improve software quality and software security. Source File of C and C++ were found to be more likely to be both buggy and vulnerable and files having no bugs fixes were having low vulnerabilities in count. Vulnerabilities types like DOS and Overflow were found to be more related to bugs. Vulnerabilities which were found to be associated with software bugs had moderate-high severity. We also provided rank lists of the files which were fixed repeatedly for bugs and vulnerabilities and which needed attention of software engineers to find out the cause of the repeated faults in the files. Google Chromium Project team can study these findings about their system and use them to take their steps to improve software quality and software security.

We found that common belief that states that security bugs are more likely to be translated into vulnerabilities is not true, in-fact security bugs were the least type of bug to be translated into vulnerabilities. According to a research on Google Chromium Project, it took two years for a bug to be converted into vulnerabilities, we found that belief was untrue, flaws can be exploited as soon as hacker or cracker finds that.

6.2 Contribution

Exploring Bug-Vulnerability Relationship upto Each Bug and Vulnerability

In the previous research on the topic, they studied overall bugs and vulnerabilities of different releases of Google Chromium Project. We studied every bug and every vulnerability seperately and explored the relationship as well as patterns that could be used to improve software security or software quality.

Bug-Vulnerability Two Way relationship Study

In the previous research, they studied bug-vulnerability relationship in the manner that if bugs can foreshadow vulnerabilities. We studied the relationship in two ways and explored vulnerability-> bug relationship also. Also, in the last research, they didnt find any significant ratio to assume that there exists relationship between Bug-Vulnerability. We found that 46% Vulnerabilities were related to bugs and 35% of the bugs were related to vulnerabilities.

Finding Patterns that can help improve Software Security and Software Quality

We found many patterns that could be used by Google Chromium Team to improve their Software Quality and Software Security.

6.3 Future Work

6.3.1 Prediction Model for associated bugs and vulnerabilities

Once a bug or vulnerability is reported, to fix that bug or vulnerability, use of prediction models is common approach. We can create a rank list of bugs and vulnerabilities which are associated with each other. Once a vulnerability is reported, it should not only be checked for similar vulnerabilities in past but it should also be checked for associated bugs which can provide us some additional files that could be fixed to remove vulnerabilities and bugs for good.

6.3.2 Studying Diffs for further exploration of relationship

We studied co occurrences of the bugs and vulnerabilities in this research, we can further explore causal relationship also. To do this, we can study the diffs in files to fix bugs and vulnerability.

6.3.3 Generalizing Relationship

We can study other software systems for bug-vulnerability relationship. If we find that same or near to this ratio exists in other software systems as well. We can generalize the relationship and can give verdict that software bugs and software vulnerabilities are related to this ratio in every software systems.

References

Abbot, C. (2017). *Security Trends in 2016: Known Vulnerabilities Are Still Dangerous.*

Agarwal, B. R., Srikant, R., & Ahmad, M. A. (1994). Fast Algorithms for Mining Association Rules. *Proceedings of the 20th VLDB Conference Santiago, Chile*, *1215*, 487–499.

Alali, A., Kagdi, H., & Maletic, J. I. (2008). What's a typical commit? A characterization of open source software repositories. *IEEE International Conference on Program Comprehension*, 182–191.

AVNER, G. (2017). *Top 10 Security Vulnerabilities of 2017.*

Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., & Cukic, B. (2017). Automated triaging of very large bug repositories. *Information and Software Technology*, *89*, 1–13.

Bozorgi, M., Saul, L. K., Savage, S., & Voelker, G. M. (2010). Beyond Heuristics : Learning to Classify Vulnerabilities and Predict Exploits. *KDD '10 Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 105–113.

Camilo, F., Meneely, A., & Nagappan, M. (2015). Do bugs foreshadow vulnerabilities? a study of the chromium project. In *Ieee/acm 12th working conference on mining software repositories* (pp. 269–279).

Dallmeier, V., & Zimmermann, T. (2007). Extraction of Bug Localization Benchmarks from History. , 433–436.

Fonseca, J., & Vieira, M. (2008). Mapping software faults with web security vulnerabilities. *Proceedings of the International Conference on Dependable Systems and Networks*, 257–266.

Gegick, M., Rotella, P., & Williams, L. (2009). Predicting attack-prone components. *Proceedings - 2nd International Conference on Software Testing, Verification, and Validation, ICST 2009*, 181–190.

Gopalakrishna, R., Spafford, E. H., Gopalakrishna, R., & Spafford, E. H. (2005). A trend analysis of vulnerabilities. *West Lafayette: Purdue University*, 1–12.

Greenberg, A. (2014). *THE 5 MOST DANGEROUS SOFTWARE BUGS OF 2014.*

Jimenez, W., Mammar, A., & Cavalli, A. (2009). Software Vulnerabilities, Prevention and Detection Methods: A Review. *Security in Model-Driven Architecture*, *215995*.

Kanwal, J., & Maqbool, O. (2012). Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, *27*(2), 397–412.

Le, T.-d., Lo, D., Goues, C., & Grunske, L. (2016). A Learning-to-Rank Based Fault Localization Approach using Likely Invariants. *Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM*, 177–188.

Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., & Zhai, C. (2006). Have Things Changed Now ? â An Empirical Study of Bug Characteristics in Modern Open Source Software. *In Proceedings of the 1st workshop on Architectural and system support for improving software dependability*, 25–33.

Liu, B., Shi, L., Cai, Z., & Li, M. (2012). Software vulnerability discovery techniques: A survey. *Proceedings - 2012 4th International Conference on Multimedia and Security, MINES 2012*, 152–156.

Liu, C., Yan, X., Fei, L., Han, J., & Midkiff, S. P. (2005). SOBER: statistical model-based bug localization. *SIGSOFT Softw. Eng. Notes*, *30*(5), 286–295.

Mcgraw, G. (2004). Software security. *IEEE Security & Privacy Magazine*, *2*(2), 80–83.

Meneely, A., Tejeda, A. C. R., Spates, B., Trudeau, S., Neuberger, D., Whitlock, K., … Davis, K. (2014). An empirical investigation of socio-technical code review metrics and security vulnerabilities. *Proceedings of the 6th International Workshop on Social Software Engineering - SSE 2014*, 37–44.

Munaiah, N., Camilo, F., Wigham, W., Meneely, A., & Nagappan, M. (2017a). Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project. *Empirical Software Engineering*, *22*(3), 1305–1347.

Munaiah, N., Camilo, F., Wigham, W., Meneely, A., & Nagappan, M. (2017b). Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project. In *Msr '15 proceedings of the 12th working conference on mining software repositories* (Vol. 22, pp. 1305–1347). Empirical Software Engineering.

Myers, G. (2004). *The Art of Software Testing, Second edition* (Vol. 15).

Nagwani, N. K., & Verma, S. (2011). Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes. *International Conference on ICT and Knowledge Engineering*, 113–117.

Neuhaus, S., Zimmermann, T., Holler, C., & Zeller, A. (2007). Predicting vulnerable software components. *Proceedings of the 14th ACM conference on Computer and communications security - CCS '07*, 529.

Özkan, S. (2012). CVEdetails.com. *Retrieved 16 (2017)*, 1–25.

Rao, S., & Kak, A. (2011). Retrieval from software libraries for bug localization. *Proceeding of the 8th working conference on Mining software repositories - MSR '11*, 43.

Shahmehri, N., Kamkar, M., & Fritzson, P. (1990). Semi-automatic bug localization in software maintenance. *Proceedings. Conference on Software Maintenance 1990*, 30–36.

Sharma, M., Kumari, M., & Singh, V. B. (2013). Understanding the meaning of bug attributes and prediction models. *Proceedings of the 5th IBM Collaborative Academia Research Exchange Workshop on - I-CARE '13*(October), 1–4.

Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2015). A time-based approach to automatic bug report assignment. *Journal of Systems and Software*, *102*, 109–122.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A., & Matsumoto, K. (2015). The impact of mislabelling on the performance and interpretation of defect prediction models. *Proceedings - International Conference on Software Engineering*, *1*, 812–823.

Xuan, J. (2012). Developer Prioritization in Bug Repositories. *Software Engineering (ICSE), 2012 34th International Conference on. IEEE*, 25–35.

Yan, M., Zhang, X., Yang, D., Xu, L., & Kymer, J. D. (2016). A component recommender for bug reports using Discriminative Probability Latent Semantic Analysis. *Information and Software Technology*, *73*, 37–51.

Zhang, W., Wang, S., & Wang, Q. (2016). KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Information and Software Technology*, *70*, 68–84.

Zheng, Z., Kohavi, R., & Mason, L. (2001). Real world performance of association rule algorithms. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, 401–406.

Appendix

Bug-Vulnerability Rules

Table 6.1: Top 100 Bug-vulnerability Rules

Rules	Support	Confidence	Count
{CVE ID= CVE-2017-15412=>{Bug ID=801488	0.004085802	0.467532468	36
{CVE ID=CVE-2015-6780=>{Bug ID=675239	0.00226989	0.666666667	20
{CVE ID= CVE-2017-15420=>{Bug ID=777419	0.001702417	0.555555556	15
{CVE ID=CVE-2015-6780=>{Bug ID=700591	0.001588923	1	14
{CVE ID= CVE-2017-15951=>{Bug ID=784761	0.001361934	0.25	12
{CVE ID=CVE-2017-5093=>{Bug ID=670135	0.001361934	0.3	12
{CVE ID=CVE-2017-5086=>{Bug ID=784761	0.001361934	0.25	12
{CVE ID=CVE-2017-5110=>{Bug ID=850078	0.001248439	0.35483871	11
{Bug ID=770946=>{CVE ID=CVE-2017-15386	0.001248439	0.169230769	11
{CVE ID= CVE-2017-0663=>{Bug ID=801488	0.001134945	0.12987013	10
{CVE ID=CVE-2014-7936=>{Bug ID=776145	0.001134945	0.357142857	10
{Bug ID=707240=>{CVE ID=CVE-2017-5110	0.001134945	0.357142857	10
{CVE ID=CVE-2015-6780=>{Bug ID=653171	0.001134945	0.333333333	10
{CVE ID=CVE-2014-7936=>{Bug ID=770946	0.001134945	0.153846154	10
{Bug ID=704675=>{CVE ID=CVE-2017-5110	0.00102145	1	9
{CVE ID=CVE-2017-5073=>{Bug ID=716474	0.00102145	0.45	9
{CVE ID= CVE-2017-5073=>{Bug ID=716474	0.00102145	0.45	9
{CVE ID=CVE-2016-5165=>{Bug ID=779156	0.00102145	0.310344828	9
{Bug ID=683731=>{CVE ID=CVE-2017-5110	0.00102145	0.333333333	9
{CVE ID= CVE-2018-6117=>{Bug ID=822465	0.000907956	1	8

Table 6.1: Top 100 Bug-vulnerability Rules

{CVE ID= CVE-2018-6032=>{Bug ID=755595	0.000907956	0.571428571	8
{CVE ID= CVE-2018-6032=>{Bug ID=700610	0.000907956	0.571428571	8
{CVE ID= CVE-2018-6032=>{Bug ID=566091	0.000907956	0.444444444	8
{Bug ID=706053=>{CVE ID=CVE-2017-5110	0.000907956	0.307692308	8
{CVE ID=CVE-2017-15386=>{Bug ID=803859	0.000907956	0.153846154	8
{CVE ID=CVE-2016-1701=>{Bug ID=838578	0.000794461	0.291666667	7
{CVE ID=CVE-2016-1690=>{Bug ID=838578	0.000794461	0.291666667	7
{CVE ID= CVE-2017-5067=>{Bug ID=815187	0.000794461	0.189189189	7
{CVE ID=CVE-2017-5061=>{Bug ID=815187	0.000794461	0.189189189	7
{CVE ID=CVE-2017-5067=>{Bug ID=815187	0.000794461	0.189189189	7
{CVE ID= CVE-2017-5061=>{Bug ID=815187	0.000794461	0.189189189	7
{Bug ID=731832=>{CVE ID=CVE-2017-5110	0.000680967	1	6
{CVE ID= CVE-2018-6117=>{Bug ID=620004	0.000680967	1	6
{CVE ID= CVE-2018-6117=>{Bug ID=620002	0.000680967	1	6
{CVE ID= CVE-2018-6117=>{Bug ID=828312	0.000680967	1	6
{CVE ID=CVE-2016-1625=>{Bug ID=718770	0.000680967	1	6
{CVE ID=CVE-2016-1671=>{Bug ID=831962	0.000680967	0.5	6
{CVE ID= CVE-2016-1671=>{Bug ID=831962	0.000680967	0.5	6
{CVE ID=CVE-2017-5110=>{Bug ID=828427	0.000680967	0.75	6
{CVE ID=CVE-2017-5104=>{Bug ID=825677	0.000680967	0.5	6
{CVE ID=CVE-2016-5132=>{Bug ID=755595	0.000680967	0.428571429	6
{CVE ID=CVE-2016-5132=>{Bug ID=700610	0.000680967	0.428571429	6
{CVE ID=CVE-2016-5132=>{Bug ID=566091	0.000680967	0.333333333	6
{Bug ID=696029=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{Bug ID=710004=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{Bug ID=772148=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{Bug ID=709571=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{Bug ID=709535=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{Bug ID=708052=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6

Table 6.1: Top 100 Bug-vulnerability Rules

{Bug ID=701345=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{CVE ID=CVE-2017-15389=>{Bug ID=750239	0.000680967	0.166666667	6
{CVE ID= CVE-2017-15390=>{Bug ID=750239	0.000680967	0.166666667	6
{Bug ID=696733=>{CVE ID=CVE-2017-5110	0.000680967	0.25	6
{Bug ID=709776=>{CVE ID=CVE-2017-5110	0.000680967	0.230769231	6
{CVE ID=CVE-2017-5093=>{Bug ID=779878	0.000680967	0.2	6
{CVE ID=CVE-2014-7936=>{Bug ID=779878	0.000680967	0.2	6
{Bug ID=657896=>{CVE ID=CVE-2017-5059	0.000680967	0.176470588	6
{CVE ID=CVE-2011-2845=>{Bug ID=657896	0.000680967	0.176470588	6
{Bug ID=687572=>{CVE ID=CVE-2017-5093	0.000680967	0.176470588	6
{CVE ID=CVE-2014-7936=>{Bug ID=687572	0.000680967	0.176470588	6
{CVE ID=CVE-2014-7936=>{Bug ID=658980	0.000680967	0.166666667	6
{CVE ID=CVE-2017-15386=>{Bug ID=670135	0.000680967	0.15	6
{CVE ID=CVE-2014-7936=>{Bug ID=670135	0.000680967	0.15	6
{CVE ID= CVE-2017-15412=>{Bug ID=643804	0.000680967	0.109090909	6
{CVE ID=CVE-2017-5093=>{Bug ID=770946	0.000680967	0.092307692	6
{CVE ID=CVE-2017-5093=>{Bug ID=776418	0.000680967	0.049180328	6
{CVE ID=CVE-2017-5104=>{Bug ID=796538	0.000680967	0.056603774	6
{Bug ID=679835=>{CVE ID=CVE-2017-5110	0.000680967	0.078947368	6
{CVE ID=CVE-2014-7936=>{Bug ID=437116	0.000567472	0.357142857	5
{CVE ID=CVE-2014-7936=>{Bug ID=493257	0.000567472	0.333333333	5
{CVE ID= CVE-2017-5067=>{Bug ID=797855	0.000567472	0.25	5
{CVE ID=CVE-2017-5061=>{Bug ID=797855	0.000567472	0.25	5
{CVE ID=CVE-2017-5067=>{Bug ID=797855	0.000567472	0.25	5
{CVE ID= CVE-2017-5061=>{Bug ID=797855	0.000567472	0.25	5
{CVE ID=CVE-2016-5132=>{Bug ID=783529	0.000567472	0.119047619	5
{CVE ID=CVE-2017-5059=>{Bug ID=803859	0.000567472	0.096153846	5
{CVE ID=CVE-2016-1625=>{Bug ID=806251	0.000453978	1	4
{Bug ID=697595=>{CVE ID=CVE-2017-5110	0.000453978	1	4

Table 6.1: Top 100 Bug-vulnerability Rules

{CVE ID= CVE-2018-6117=>{Bug ID=771878	0.000453978	1	4
{CVE ID= CVE-2018-6117=>{Bug ID=617364	0.000453978	1	4
{Bug ID=766973=>{CVE ID=CVE-2017-5110	0.000453978	1	4
{Bug ID=770303=>{CVE ID=CVE-2017-5110	0.000453978	1	4
{CVE ID= CVE-2018-6117=>{Bug ID=751434	0.000453978	1	4
{CVE ID= CVE-2018-6117=>{Bug ID=827579	0.000453978	1	4
{CVE ID=CVE-2017-5110=>{Bug ID=742329	0.000453978	1	4
{CVE ID=CVE-2017-5080=>{Bug ID=736944	0.000453978	1	4
{CVE ID=CVE-2015-6780=>{Bug ID=640851	0.000453978	1	4
{CVE ID= CVE-2018-6032=>{Bug ID=777275	0.000453978	1	4
{CVE ID= CVE-2018-6117=>{Bug ID=798906	0.000453978	1	4
{CVE ID=CVE-2016-5190=>{Bug ID=623606	0.000453978	0.142857143	4
{CVE ID=CVE-2015-6785=>{Bug ID=625945	0.000453978	1	4
{CVE ID= CVE-2018-6117=>{Bug ID=808182	0.000453978	1	4
{CVE ID= CVE-2018-6117=>{Bug ID=649551	0.000453978	1	4
{CVE ID=CVE-2016-1625=>{Bug ID=625161	0.000453978	1	4
{CVE ID=CVE-2016-1616=>{Bug ID=575789	0.000453978	1	4
{CVE ID= CVE-2016-5133=>{Bug ID=716296	0.000453978	0.333333333	4
{CVE ID=CVE-2016-1616=>{Bug ID=590941	0.000453978	1	4
{CVE ID=CVE-2016-5133=>{Bug ID=716296	0.000453978	0.333333333	4
{CVE ID= CVE-2018-6117=>{Bug ID=812035	0.000453978	1	4

Rank list of Buggy Files

Table 6.2: Top 100 Buggy Files Rank List

Files	Bug Count
histograms.xml	61
about_flags.cc	60
generated_resources.grd	54
enums.xml	48
flag_descriptions.cc	39

Table 6.2: Top 100 Buggy Files Rank List

compiled_resources2.gyp	37
flag_descriptions.h	36
DEPS	31
payment_request.cc	27
extensions_ui.cc	26
detail_view.html	23
manager.html	21
md_extensions_strings.grdp	21
payment_sheet_view_controller.cc	21
location_bar_view.cc	20
tab.cc	20
tab_strip.cc	20
toolbar.html	20
browser_non_client_frame_view_ash.cc	19
shipping_address_editor_view_controller.cc	19
chrome_content_browser_client.cc	18
chrome_features.cc	18
local_ntp.js	18
manage_passwords_bubble_view.cc	18
manager.js	18
browser_view.cc	17
credit_card_editor_view_controller.cc	17
credit_card_editor_view_controller_browsertest.cc	17
item.html	16
local_ntp.css	16
payment_request.h	16
chrome_features.h	15
item.js	15
payment_request_state.cc	15

Table 6.2: Top 100 Buggy Files Rank List

bookmark_app_navigation_throttle_browsertest.cc	14
chrome_switches.cc	14
chrome_switches.h	14
location_bar_view.h	14
md_settings_localized_strings_provider.cc	14
omnibox_result_view.cc	14
payment_request_browsertest_base.cc	14
payment_request_views_util.cc	14
shipping_address_editor_view_controller_browsertest.cc	14
autofill_experiments.cc	13
bookmark_app_navigation_throttle.cc	13
bookmark_bar_view.cc	13
detail_view.js	13
layout_constants.cc	13
payment_method_view_controller.cc	13
settings_strings.grdp	13
app.html	12
autofill_experiments.h	12
autofill_manager.cc	12
credit_card_editor_view_controller.h	12
error_page.html	12
passwords_section.html	12
payment_request_browsertest_base.h	12
render_frame_host_impl.cc	12
theme_resources.grd	12
autofill_interactive_uitest.cc	11
browser.cc	11
chrome_tests_unit.gypi	11
contact_info_editor_view_controller.cc	11

Table 6.2: Top 100 Buggy Files Rank List

cr_extensions_browsertest.js	11
error_page.js	11
item_list.html	11
local_ntp_browsertest.cc	11
security_state.cc	11
shipping_address_editor_view_controller.h	11
toolbar.js	11
bookmark_app_helper.cc	10
browser_non_client_frame_view_ash_browsertest.cc	10
chrome_browser.gypi	10
chrome_tests.gypi	10
editor_view_controller.cc	10
hosted_app_browsertest.cc	10
keyboard_shortcuts.html	10
local_ntp_source.cc	10
omnibox_edit_model.cc	10
omnibox_view_views.cc	10
page_info.cc	10
page_info_bubble_view.cc	10
password_autofill_agent.cc	10
tab.h	10
tab_unittest.cc	10
actions.xml	9
autofill_metrics_unittest.cc	9
autofill_strings.grdp	9
browser_non_client_frame_view_ash.h	9
browser_resources.grd	9
chrome_browser_ui.gypi	9
extension_detail_view_test.js	9

Table 6.2: Top 100 Buggy Files Rank List

hosted_app_button_container.cc	9
most_visited_single.js	9
navigation_handle_impl.cc	9
omnibox_popup_contents_view.cc	9
payment_request_dialog_view.cc	9
payment_request_spec.cc	9

Rank List of Vulnerable Files

Table 6.3: Top 100 Vulnerable Files

Files	**Vulnerability Count**
render_frame_impl.cc	78
manifest.json	36
enums.xml	22
payment_request.cc	21
navigator_impl.cc	21
md_settings_localized_strings_provider.cc	13
payment_request_browsertest_base.cc	12
payment_request_browsertest_base.h	12
PaymentRequestImpl.java	10
payment_request.h	10
LocalFrame.cpp	10
layer_tree_host.cc	8
layer_tree_host.h	8
layer_tree_host_impl.cc	8
chrome_tests_unit.gypi	8
autofill_agent.cc	6
test.js	6
chrome_content_browser_client.h	6
site_per_process_browsertest.cc	5

Table 6.3: Top 100 Vulnerable Files

actions.xml	5
chrome_browser_ui.gypi	5
passwords_section.html	4
chrome_tests.gypi	4
navigation_handle_impl_browsertest.cc	4
save_card_bubble_controller_impl.cc	4
ffmpeg_regression_tests.cc	4
test_chrome_payment_request_delegate.cc	3
test_chrome_payment_request_delegate.h	3
ssl_browsertest.cc	3
main.js	3
save_card_bubble_controller_impl_unittest.cc	3
languages_page.js	3
render_view_impl.cc	3
content_switches.h	3
content_browser_client.cc	3
content_browser_client.h	3
crw_web_controller.mm	3
views.gyp	3
cursor_impl.cc	3
indexed_db_cursor.cc	3
ContainerNode.cpp	3
HTMLFormControlElement.cpp	3
fieldtrial_testing_config.json	2
main.html	2
browser_view.cc	2
ukm.xml	2
OWNERS	2
idn_spoof_checker.h	2

Table 6.3: Top 100 Vulnerable Files

alexa_domains.list	2
alexa_skeletons.gperf	2
make_alexa_top_list.py	2
render_frame_host_delegate.h	2
render_view_impl.h	2
journey_logger_unittest.cc	2
website_settings_popup_view.h	2
pref_names.h	2
gles2_cmd_decoder.cc	2
zoom_bubble_view.cc	2
zoom_bubble_view.h	2
WebGL2RenderingContextBase.h	2
VisibleUnits.cpp	2
color-no-event-during-detach.html	2
input-color-choose-default-value-after-set-value.html	2
input-color-onchange-event-expected.txt	2
input-color-onchange-event.html	2
HTMLTextFormControlElement.cpp	2
HTMLTextFormControlElement.h	2
ColorInputType.cpp	2
elements.cc	2
webstore_inline_installer_browsertest.cc	2
debugger_api.cc	2
render_thread_impl.h	2
objects.h	2
objects.cc	2
script_context.cc	2
extension_bindings_apitest.cc	2
UrlBar.java	2

Table 6.3: Top 100 Vulnerable Files

indexed_db_callbacks.cc	2
SkArenaAlloc.h	2
IDS_SETTINGS_CAN_MAKE_PAYMENT_TOGGLE_LABEL.png.sha1	1
cr_dialog.html	1
autofill_metrics.cc	1
autofill_metrics_unittest.cc	1
cr_drawer.js	1
password_form_manager.cc	1
chrome_features.h	1
autofill_manager.h	1
cr_action_menu.html	1
autofill_experiments.h	1
autofill_experiments_unittest.cc	1
answer_when_is.icon	1
md_extensions_strings.grdp	1
chrome_web_ui_controller_factory.cc	1
password_autofill_manager_unittest.cc	1
password_manager_metrics_util.h	1
security_state_unittest.cc	1
hosted_app_menu_button.h	1
omnibox_popup_contents_view_browsertest.cc	1
md_bookmarks_focus_test.js	1