


Investigating Vulnerability-Fixing Commits

Vinícius Almeida   [Universidade Federal de Pernambuco | vsa@cin.ufpe.br]

Rodrigo Andrade  [Universidade Federal do Agreste de Pernambuco | rodrigo.andrade@ufape.edu.br]

 Centro de Informática, Universidade Federal de Pernambuco, Av. Jorn. Aníbal Fernandes, Cidade Universitária, Recife, PE, 50740-560, Brazil.

Received: 06 July 2024 • Accepted: 24 February 2025 • Published: 15 May 2025

Abstract An insecure software can cause severe damage to the user experience and privacy. Therefore, developers should prevent software vulnerabilities. However, detecting such problems is expensive and time-consuming. To address this issue, researchers propose vulnerability datasets that facilitate the investigation of their properties. In this regard, we investigate one of these datasets to better understand the vulnerabilities, their corrections, the authors involved, and the properties of the correction commits. Our results indicate that some vulnerabilities require many patches to solve. Furthermore, among the projects included in the target dataset, the Chromium project is the most affected by these vulnerabilities. We also find that in most cases correction commits are small in terms of the number of files and lines affected. Additionally, the authors of the corrections are mostly not new to the files that need fixing. Finally, we find that most corrections involve changes that affect other developers and rarely affect the developer who introduced the problem. Therefore, corrections are usually made by other developers rather than by those who introduced the problem. We believe that our findings can help developers resolve vulnerabilities with fewer resources, such as time, budget, and manpower.

Keywords: Vulnerability. Datasets. Commits. Common Vulnerabilities and Exposures.

1 Introduction

Security is a major concern for software developers. Insecure software is costly to maintain and harmful to user experience and privacy [Bosu *et al.*, 2014]. Therefore, properly addressing security is crucial. To build secure software, developers aim to prevent software vulnerabilities [Meneely *et al.*, 2013]. These vulnerabilities represent instances of errors in the specification, development, or configuration of software in such a way that their execution might violate a security policy. Software vulnerabilities could have consequences that extend beyond system failures [Krsul, 1998]. Exploiting these vulnerabilities could compromise user privacy and have disastrous effects [Fan *et al.*, 2020].

Unfortunately, finding software vulnerabilities requires expertise in the specific system and software security [Allen *et al.*, 2006; McGraw *et al.*, 2008]. Consequently, identifying and fixing vulnerabilities is expensive and time-consuming [Ponta *et al.*, 2019]. To overcome this issue, researchers propose a range of solutions, such as tools to automatically detect certain types of errors [Perl *et al.*, 2015; Graf *et al.*, 2013] or datasets of known vulnerabilities [Fan *et al.*, 2020; Gkortzis *et al.*, 2018; Ponta *et al.*, 2019]. In particular, these datasets consist of a collection of vulnerability records typically linked to an existing vulnerability database, such as the Common Vulnerabilities and Exposures (CVE) [The MITRE Corporation, 2023]. They enable developers and researchers to investigate various properties related to vulnerabilities and their resolutions.

In this study, our primary goal is to investigate the characteristics of commits that address vulnerabilities. To achieve this, we formulate seven research questions aimed at identify-

ing the vulnerabilities with the highest number of patches to solve, understanding the complexity of fix patches in terms of the number of lines and files altered, assessing the experience of authors handling these corrections, and determining whether the fixes impact code written by other developers or by the authors themselves.

These findings could help developers understand the complexity of vulnerability-fixing patches in lines of code or in the number of files. Also, they may help project managers assign developers to fix vulnerabilities based on developers' experience in the code and the project, or even in the vulnerability itself. In addition, this work provides future work on the development of automated tools to help project managers with these tasks.

To answer these research questions, we measure a set of metrics considering vulnerability-fixing commits present in the Big-Vul dataset [Fan *et al.*, 2020]. These commits are scattered across four open-source projects: Chromium [The Chromium Projects, 2023], Linux [The Linux Kernel Archives, 2023], ImageMagick [ImageMagick, 2023], and FFmpeg [FFmpeg, 2023].

Our findings indicate that the vulnerabilities with the highest number of patches to fix in the Big-Vul dataset are specific to the projects in which they occur. This is because most VFCs are derived from Common Vulnerabilities and Exposures (CVEs), which are tied to the context of individual software projects. For example, CVE-2021-2875¹ highlights a vulnerability in the PDF functionality of Google Chrome, illustrating how these vulnerabilities often relate to project-

¹<https://cve.mitre.org/cgi-bin/cvename.cgi?name=2012-2875>

specific features or components. Additionally, we find that vulnerability fix patches typically do not affect a large number of files or lines of code. Moreover, our results demonstrate that the authors handling vulnerability fixes are experienced within the projects they are working on, and they are not new effective authors of the code they fix. Lastly, we observe that the corrections are generally carried out by authors other than those who originally introduced the vulnerabilities.

In this way, the contributions of this work help developers understand the characteristics of vulnerability fixes. They also assist in defining the right developers for these fixes, using factors such as experience or even whether the developer contributed to the introduction of the vulnerability or not. Moreover, we make the scripts and the data from the studied projects available in our online Appendix [Almeida and Andrade, 2024].

The remainder of this work is organized as follows. Section 2 presents the key concepts necessary for a comprehensive understanding of this research. Furthermore, Section 3 describes our research method. Next, in Section 4, we explain our assessment as well as its results. Following that, in Section 5, we discuss related work. Finally, in Section ??, we provide the conclusion of this work.

2 Background

This section presents the concepts necessary for an understanding of this work. We begin by defining vulnerabilities in Section 2.1. Moreover, in Section 2.2, we explain concepts of vulnerability datasets and, in particular, the one we use in this work, which is called Big-Vul [Fan *et al.*, 2020].

2.1 Vulnerability

A software vulnerability is an instance of an error in the specification, development, or configuration of software such that its execution can violate a security policy [Krsul, 1998]. Therefore, a vulnerability can be just a simple coding mistake, but it can lead to catastrophic failures that compromise the confidentiality, integrity, and availability of the system [Meneely *et al.*, 2014].

In the context of our work, there are two types of commits related to vulnerabilities. First, the Vulnerability-Contributing Commit (VCC) represents commits that contribute to the introduction of vulnerabilities in a software project [Meneely *et al.*, 2013; Bosu *et al.*, 2014]. Second, Vulnerability-Fixing Commit (VFC) refers to commits that remove or correct a vulnerability that was introduced in an VCC [Wan, 2019].

In this context, developers often report identified vulnerabilities in repositories such as the Common Vulnerabilities and Exposures (CVE) [The MITRE Corporation, 2023] or the National Vulnerability Database [National Institute of Standards and Technology. National Vulnerability Database (NVD), 2021]. Table 1 provides an example of a vulnerability registered in the CVE².

Table 1. CVE-2012-2390 properties

Description	Memory leak in mm/hugetlb.c in the Linux kernel before 3.4.2 allows local users to cause a denial of service (memory consumption or system crash) via invalid MAP_HUGETLB mmap operations.
Confidentiality impact	None
Integrity impact	None
Availability impact	Complete
Type	Denial of Service (DoS)
Access complexity	Low

The CVE-2012-2390 vulnerability was identified by the Red Hat Linux Team³ in June 2012. It is related to a problem in a Linux kernel file that malicious developers could exploit to cause a Denial of Service (DoS) [Wood and Stankovic, 2002]. If successfully exploited, the correct exploit of this vulnerability could lead to complete unavailability of the executing Linux kernel. Therefore, in Section 2.2, we explain how researchers and developers use datasets to gather information about vulnerabilities and their corrections.

2.2 Vulnerability datasets

To investigate the issue of a VCC, researchers propose a number of solutions, such as tools for automatically detecting certain types of vulnerabilities [Perl *et al.*, 2015; Graf *et al.*, 2013] and datasets of known vulnerabilities [Fan *et al.*, 2020; Gkortzis *et al.*, 2018; Ponta *et al.*, 2019]. In particular, a dataset comprises a collection of vulnerability records typically linked to an existing vulnerability database, such as the Common Vulnerabilities and Exposures (CVE) [The MITRE Corporation, 2023]. These datasets enable developers and researchers to investigate various properties related to vulnerabilities.

In this way, we could analyze the records in a dataset to gather useful information, such as the vulnerability identification number, classification, publication date, or link to the fix commit on GitHub [Fan *et al.*, 2020; Liu *et al.*, 2020]. For example, given the link to the fix commit on GitHub, we could gather additional useful information about the commit, such as its date, author, commit message, modified files and lines of code, or programming language [Fan *et al.*, 2020; Liu *et al.*, 2020].

Furthermore, we can use this information to better understand how developers address a specific vulnerability in a project. For example, we could analyze the sociotechnical context in which these vulnerabilities are fixed [Meneely and Williams, 2012]. Consequently, various datasets are available for this purpose [Fan *et al.*, 2020; Gkortzis *et al.*, 2018; Liu *et al.*, 2020; Perl *et al.*, 2015; Ponta *et al.*, 2019]. In this work, we use the Big-Vul [Fan *et al.*, 2020]. We provide further details in Section 3.2.

3 Study Settings

To begin our study, we analyze five different researches about Vulnerability Datasets. In the first research [Liu *et al.*,

²<https://www.cve.org/CVERecord?id=CVE-2012-2390>

³<https://www.redhat.com/pt-br/authors/red-hat-enterprise-linux-team>

2020], the authors select five open-source projects and investigate public vulnerability data sources, including the National Vulnerability Database (NVD) [National Institute of Standards and Technology. National Vulnerability Database (NVD), 2021], Bugzilla reports [The Bugzilla Guide, 2024], security bulletins, and GitHub commits. In the second study [Perl *et al.*, 2015], the authors develop a code analysis tool that detects suspicious commits using an SVM-based detection model. They also provide a large VCC database with the results. The third research [Ponta *et al.*, 2019] provides a manually curated dataset on vulnerability fixes called Vulas, with more than 1,200 commits to vulnerability fixing from 205 different open-source Java projects. However, we believe that the sample of the dataset is small given the number of projects. The fourth research [Gkortzis *et al.*, 2018] introduces the VulinOSS, which is a vulnerability dataset stored in a database with information such as CVE, CWE, and metrics encompassing 156 projects. The last research [Fan *et al.*, 2020] introduces the Big-Vul, which is a C/C++ vulnerability dataset containing information such as CVE IDs, CVE severity scores, CVE summaries, code changes, project name, etc. It includes 348 projects in a CSV file with commit hashes before and after the fix. We selected the Big-Vul dataset over the previous studies cited since it covers more projects and vulnerabilities than other datasets [Ponta *et al.*, 2019; Gkortzis *et al.*, 2018]. Also, the Big-Vul offers its data in a single CSV file. Therefore, we select the Big-Vul as the dataset for our work. We provide further details in Section 3.2.

The remainder of this section is organized as follows. In Section 3.1, we explain our research questions and metrics. In addition, in Section 3.2, we discuss the vulnerability dataset that we select for this work, namely Big-Vul, and we explain the target software projects that we use in this work. Finally, we explain how we conduct data collection and analysis to draw our conclusions in Section 3.3.

3.1 Goal-Question-Metric

To better drive our work, we adopt the Goal-Question-Metric (GQM) approach [Basili *et al.*, 1994]. Our goal is to investigate the characteristics of commits that address vulnerabilities in open-source software. To achieve this goal, we define seven research questions. We aim to answer these questions with the help of seven software metrics. Table 2 summarizes our GQM approach.

As shown in Table 2, **RQ1** is important to understand which vulnerabilities have the highest number of patches required to solve in software projects. Developers could use such information to focus their efforts on searching for certain types of vulnerabilities, and decrease the number of patches required to fix the CVE. To answer **RQ1**, we use the Number of Patches to Solve (NPS) metric, which is the number of occurrences of a specific vulnerability, either in a project or in a dataset. Therefore, our goal is to identify the vulnerabilities with the highest number of patches that might enable developers to be aware of them and take the necessary measures to solve them in fewer patches.

Moreover, the answers to **RQ2** and **RQ3** might help developers understand the size of vulnerability fix patches. To answer them, we use two metrics: Code Churn [Meneely and

Williams, 2012] and File Churn. The former measures the number of lines modified in a file in a given commit. For example, the sum of added and removed lines. On the other hand, we define the latter as the number of files altered by the author. In other words, File Churn represents the number of files added, removed, or modified in a commit. Therefore, answering these research questions can help developers evaluate beforehand the effort required to fix vulnerabilities in the number of lines and files changed.

Regarding **RQ4** and **RQ7**, our aim is to investigate two aspects: first, whether a vulnerability-fixing commit author is effectively new to the corresponding code and second, whether an author has prior experience in fixing vulnerabilities within a specific project. To answer both research questions, we use the New Effective Author (NEA) and the Number of Commits Before (NCB) metrics. NEA is a nominal ‘Yes’ when an author has previously committed to a given file or a ‘No’ otherwise [Meneely *et al.*, 2013]. In contrast, we define the NCB metric as the number of commits by an author in a project before a given commit. Thus, answers to these research questions may help developers better define whether an author should be assigned to fix the vulnerability based on his experience with the vulnerable code or in the project overall.

Additionally, **RQ5** is important to provide evidence that the author of a vulnerability fix is the same author who introduced the issue previously. For that purpose, we define the Percentage of Self Churn (PSC). It represents the percentage of modified lines that were written by the commit author out of the total number of altered or removed lines. For example, if there are only line additions in a given file, the PSC is zero because no lines were removed or altered; if five lines were modified and only three of these lines were written by the commit author, then the PSC is 60%, as 60% of the removed or altered lines were written by the commit author. This metric is based on the Percentage of Interactive Churn (PIC) metric Meneely and Williams [2012].

At last, **RQ6** aims to determine whether Vulnerability-Fixing Commits commonly affect code written by other developers. This could indicate whether vulnerabilities are introduced through the contributions of numerous developers, a few developers, or just one developer. In this context, we use the Number of Affected Authors (NAA) metric [Meneely *et al.*, 2013], which represents the number of distinct authors in addition to the commit author whose lines were affected by a given commit.

3.2 Vulnerability Dataset

In this work, we use the Big-Vul vulnerability dataset [Fan *et al.*, 2020]. It is a large dataset that contains records of vulnerabilities from open-source GitHub projects written in C/C++. Additionally, this dataset associates vulnerability occurrences with their corresponding Common Vulnerabilities Exposures (CVE) [The MITRE Corporation, 2023] identification. The CVE project consists of a list of records in which each vulnerability is associated with an identification number, a description, and at least one public reference where the vulnerability was discovered. Anyone can contribute to the list by updating a vulnerability record found in their project.

Table 2. Our GQM approach

Goal	
<i>Purpose</i>	To investigate commits that address vulnerabilities
<i>Issue</i>	for open-source software
<i>Object</i>	from a developer viewpoint
<i>Viewpoint</i>	
Questions and Metrics	
RQ1- What are the five CVEs that require the highest number of patches to be resolved?	-Number of Patches to Solve (NPS)
RQ2- How many lines do the Vulnerability-Fixing Commits patches change?	-Code Churn
RQ3- How many files do the Vulnerability-Fixing Commits patches change?	-File Churn
RQ4- Are the authors of Vulnerability-Fixing Commits new to the code that they are changing?	-New Effective Author (NEA)
RQ5- Do the Vulnerability-Fixing Commits change code written by the same author of the Vulnerability-Contributing Commit?	-Percentage of Self Churn (PSC)
RQ6- Do the Vulnerability-Fixing Commits affect code written by many other developers?	-Number of Affected Authors (NAA)
RQ7- How experienced are the authors of Vulnerability-Fixing Commits in the project?	-Number of Commits Before (NCB)

In summary, the Big-Vul comprises 4,432 code commits containing fixes for 3,754 code vulnerabilities across 348 different open-source projects, encompassing 91 different types of vulnerability. Additionally, the Big-Vul dataset enables analysis of different vulnerability characteristics and the corresponding code changes that fix these vulnerabilities. It could also improve vulnerability detection and resolution [Fan *et al.*, 2020]. In addition to CVE identification, Big-Vul records contain details such as vulnerability availability, integrity, and confidentiality, publication date, summary, vulnerability classification, commit hash, and commit message that resolves the issue. Further, it includes information about the modified files, programming language (C or C++), project name, and the commit hashes before and after the vulnerability fix.

This information is available as illustrated in Table 3. Each row represents a tuple of feature and its description. For example, CVE ID is the unique identifier for the vulnerability in the CVE database [The MITRE Corporation, 2023], while the CVE Page provides the link to the vulnerability on the CVE website. The Project, Commit ID, and Reference Link show the project where the vulnerability was found, the commit that fixes it in the project, and the link to the public reference for the fix, respectively.

To answer our seven research questions, we select four software projects within Big-Vul: Chromium [The Chromium Projects, 2023], Linux [The Linux Kernel Archives, 2023], ImageMagick [ImageMagick, 2023], and FFmpeg [FFmpeg, 2023]. In this context, we analyze their commits, the lines of code that affected those commits, and the developers who authored those commits. In this way, we limit our options to publicly hosted open-source projects on GitHub with access to the entire project code, commit history, and commit authors.

First, Chromium is an open-source web browser developed by Google, which also serves as the foundation for browsers from other companies. This makes Chromium the most popular web browser today [W3Schools, 2023]. Chromium is written in the C programming language and has a public mirror repository on GitHub with more than a million commits and over 2,900 contributors [The Chromium Repository on GitHub, 2023]. It also has the highest number of vulnerability occurrences in Big-Vul, with 1,518 records.

Second, Linux is an open source operating system that is used on desktops, servers, and mobile devices. It is written in the C programming language. Linux has a public repository on GitHub with almost 15,000 contributors and over a million commits [The Linux Repository on GitHub, 2023]. Similarly to Chromium, several companies contribute to its development. In 2017, private companies, such as Red Hat and Intel, were doing over 85% of Linux's development [The Linux Foundation, 2017]. Additionally, Linux has 927 records of vulnerabilities in Big-Vul.

Third, ImageMagick is also open source software; it is a suite used for editing and manipulating digital images. ImageMagick features a command line interface that allows the execution of complex image processing tasks, as well as APIs to integrate its functions into other programs [ImageMagick, 2023]. It is written in C and is publicly available on GitHub with 149 contributors and more than 20,000 commits [The ImageMagick Repository on GitHub, 2023]. Moreover, ImageMagick has 189 vulnerability records in Big-Vul.

Finally, FFmpeg is an open source multimedia tool set capable of performing various operations on multimedia content, such as images, videos, and audios [FFmpeg, 2023]. FFmpeg is also written in C and supported on most popular platforms, such as Linux, Windows, and MacOS. FFmpeg

Table 3. Big-Vul Features

Feature	Description
access_complexity	Reflects the complexity of the attack required to exploit the software feature misuse vulnerability
authentication_required	If authentication is required to exploit the vulnerability
availability_impact	Measures the potential impact to availability of a successfully exploited misuse vulnerability
commit_id	Commit ID in code repository, indicating a mini-version
commit_message	Commit ID in code repository, indicating a mini-version
confidentiality_impact	Measures the potential impact on confidentiality of a successfully exploited misuse vulnerability
cwe_id	Common Weakness Enumeration ID
cve_id	Common Weakness Enumeration ID
cve_page	CVE Details web page link for that CVE
summary	CVE Details web page link for that CVE
score	The relative severity of software flaw vulnerabilities
files_changed	All the changed files and corresponding patches
integrity_impact	Measures the potential impact to integrity of a successfully exploited misuse vulnerability
version_after_fix	Mini-version ID after the fix
version_before_fix	Mini-version ID before the fix
lang	Project programming language
project	Project programming language
publish_date	Publish date of the CVE
ref_ink	Reference link in the CVE page
update_date	Update date of the CVE
vulnerability_classification	Vulnerability type

is also publicly available on GitHub with more than 1,300 contributors and surpassing 100,000 commits [The FFmpeg Repository on GitHub, 2023]. This project has 84 records in Big-Vul.

3.3 Data Collection and Analysis

For the purpose of collecting data from the Big-Vul dataset, we formulate three queries. The first query aims to retrieve information regarding vulnerabilities, selecting their IDs and counting their occurrences in the Big-Vul dataset. The second query searches for the amount of VFCs per project, sorting the results from the highest to the lowest amount of VFCs. On the other hand, the last query retrieves the hashes of available VFCs in the Big-Vul dataset and organizes them by project.

Additionally, to obtain the metric values (Table 2), we execute the following procedure. First, we clone the selected target projects (Chromium, Linux, ImageMagick, and FFmpeg) to our machine. Then, we write and execute scripts to calculate each metric. These scripts use Git commands to retrieve information related to VFCs and their authors. Afterwards, we develop and run another set of scripts that query the GitHub REST API [GitHub REST API, 2021] to retrieve information about the experience of the VFC authors in each selected system. Finally, we create and execute scripts to generate charts for each metric of every project. We provide the queries and additional scripts in our online Appendix [Almeida and Andrade, 2024].

4 Evaluation

In this section, we present our assessment. Sections 4.1 to 4.7 answer our seven research questions. Following that, in Section 4.8, we discuss the answers and their consequences. Last but not least, in Section 4.9, we detail the threats to the validity of our study.

4.1 Number of patches to solve

For **RQ1**, we aim to determine the five vulnerabilities that have the highest number of patches to solve. Particularly, for this research question, we take into account all projects included in the Big-Vul dataset. Table 4 displays the top five vulnerabilities with the highest number of occurrences in Big-Vul. To gather this information, we search the dataset by selecting distinct CVEs along with their number of occurrences. Additionally, we extract the projects where these vulnerabilities occur and their vulnerability descriptions. Thus, the first column of the table contains the CVE Identifier, which is a unique ID referencing a known vulnerability in the CVE [The MITRE Corporation, 2023]. The second column, NPS, corresponds to the number of vulnerability occurrences. The Project column represents the project name where the vulnerability occurred, and the Description column shows a synopsis of the CVE vulnerability [The MITRE Corporation, 2023].

The first vulnerability with the highest number of patches to solve is CVE-2012-2827, which appears in the Chromium

Table 4. Top 5 CVEs with highest number of patches to solve

CVE ID	NPS	Project	Description
CVE-2012-2875	15	Chromium	Multiple unspecified vulnerabilities in the PDF functionality of Google Chrome before version 22.0.1229.79 allow remote attackers to potentially exploit the system via a crafted document.
CVE-2015-1265	14	Chromium	Multiple unspecified vulnerabilities in Google Chrome before version 43.0.2357.65 allow attackers to cause a denial of service or potentially have other impacts via unknown vectors.
CVE-2013-0892	7	Chromium	Multiple unspecified vulnerabilities in the IPC layer of Google Chrome before version 25.0.1364.97 on Windows and Linux, and before 25.0.1364.99 on Mac OS X, allow remote attackers to cause a denial of service or potentially have other impacts via unknown vectors.
CVE-2017-6903	7	OpenJK	In ioquake3 and other id Tech 3 engine forks before 2017-03-14, insufficient content restrictions in the auto-downloading feature allow malicious files to load as native code DLLs, potentially overriding user configurations and enabling sandbox escape.
CVE-2011-3110	6	Chromium	The PDF functionality in Google Chrome before version 19.0.1084.52 allows remote attackers to cause a denial of service or other impacts via out-of-bounds write operations.

project. This vulnerability affects the PDF reading function of the browser, allowing malicious PDF files to launch remote attacks that could cause immeasurable harm to the browser user [CVE-2012-2827, 2023].

The second and third vulnerabilities with the highest number of patches found in Big-Vul are CVE-2015-1265 and CVE-2013-0892, both of which occur in the Chromium project. These vulnerabilities enable attacks that could lead to application service failures, resulting in program crashes or even other unknown impacts [CVE-2015-1265, 2023; CVE-2013-0892, 2023].

The fourth vulnerability with the highest number of patches is CVE-2017-6903. Unlike the others, this vulnerability occurs in the OpenJK project, which maintains the engines to run certain old games [OpenJK, 2023]. This vulnerability allows malicious files to be downloaded and alters user settings, as well as modifies unwanted DLL code, leading to game crashes [CVE-2017-6903, 2023].

The last vulnerability with the highest number of patches analyzed is CVE-2011-3110, which also occurred in the Chromium project. Similarly to the first vulnerability, it also affects the Chromium PDF function. This vulnerability allowed remote attacks through malicious PDFs that caused a service failure in the application, resulting in the shutdown of the application due to write operations outside the designated memory space for Chromium [CVE-2011-3110, 2023].

Last but not least, we answer **RQ1** stating that the vulnerabilities with CVE identifications CVE-2012-2827, CVE-2015-1265, CVE-2013-0892, CVE-2017-6903, and CVE-2011-3110 present the highest number of occurrences within the Big-Vul dataset, that is, the highest number of patches to solve. However, these vulnerabilities are project-specific. Consequently, developers may not be able to use the answers to **RQ1** to proactively solve these vulnerabilities in their own projects with fewer patches, as it is unlikely that the same vulnerabilities would manifest in the same way in a different project. Nevertheless, these results could still help developers solve similar vulnerabilities in other projects quicker and

with fewer patches.

4.2 Number of Altered Lines

For **RQ2**, we determine how many lines of code developers alter in VFC patches. To answer this question, we calculate the Code Churn metric by iterating through the VFCs of each project using their *commit_id* found in the Big-Vul dataset. This can be accomplished by retrieving the *files_changed* feature from the Big-Vul dataset and parsing its data or by iterating through the projects' commits to extract the data. We opt for the latter approach.

To achieve this, we develop a script that works as follows.

1. Run the `git checkout -f ${hash}` to navigate to the desired commit.
2. Run the `git diff --name-only HEAD~1` command to retrieve the files changed between the checked-out commit and the immediate commit before it.
3. For each file in a commit, run the `git log -1 -p -U0 -- "${file}"` with the `grep` command to filter the number of lines added or removed.
4. Then, we count the number of added and removed lines.
5. Finally, we sum the total number of added line with the total number of removed lines resulting in the Code Churn.

Table 5 illustrates the values of the Code Churn metric for the VFCs of the selected projects. The resulting table has five columns, and the first represents the project. The second column, **Mean**, represents the sum of all Code Churn values divided by the number of values for each VFC in the project. The next column, **Median**, shows the central value among the Code Churn values of each VFC in the project. We also calculate the **Trimmed Mean** by removing 10% of the data extremes. In this context, the **Trimmed Mean** is interesting because the values of **Mean** and **Median** are significantly different, which indicates the presence of outliers in the set

of Code Churn values for each project. Lastly, the column **#VFCs** displays the total quantity of VFCs for each project.

Table 5. Code Churn metric

Project	Mean	Median	Trimmed Mean	# VFCs
Chromium	222.07	27.50	53.82	1518
Linux	30.77	8.00	13.82	927
ImageMagick	36.48	6.00	9.92	189
FFmpeg	9.00	5.00	6.38	84

As shown in Table 5, the Chromium project has a **Mean** of 222.07 additions and deletions of code in VFCs. This may indicate that vulnerability-fixing patches in the project often require a significant amount of code changes. However, when looking at the **Median** value, we notice that it is much lower than the **Mean**: 27.5. This indicates that the Mean value does not reflect the common number of lines changed in a VFC for the Chromium project. In other words, we can only say that there are some fixes that do require many lines. However, it is not common. Moreover, the **Trimmed Mean** for Chromium is 53.82. Therefore, it indicates that the number of changes tends to be lower than the **Mean** suggests.

Different from Chromium, the **Mean** (30.77) and the **Median** (8.00) for Linux VFCs indicate that its fix patches require a lower number of changes. Furthermore, since the **Mean** is higher than the **Median**, we can infer that the outlier VFCs have an impact on the results. In this context, the **Trimmed Median**, which is 13.82, reassures that Linux has smaller VFCs than Chromium.

Regarding the ImageMagick project, the Code Churn metric presents slightly lower values than those of Linux. This indicates that ImageMagick undergoes fewer code changes in vulnerability fixes.

Furthermore, the FFmpeg project presents similar **Median** and **Trimmed Mean** values to ImageMagick. However, interestingly, its **Mean** (9.00) represents roughly 25% of the ImageMagick result. This indicates that there are few outlier VFCs for FFmpeg.

At last, we answer **RQ2** by stating that vulnerability-fixing patches usually require only a few lines of code changes for the selected projects. However, the size of VFCs might vary across different projects. Consequently, we also assert that once the vulnerability and the vulnerable code segment have been identified, fixing the vulnerability should not require a significant number of lines of code changes.

4.3 Number of Altered Files

Concerning **RQ3**, we aim to measure how many files are affected by VFCs. Therefore, we calculate the File Churn metric for the selected projects. This can also be achieved by fetching the *files_changed* feature from the Big-Vul dataset or by iterating through the projects' *commit_id* to extract the necessary information. We opt for the latter approach, since we already have similar scripts from **RQ2**.

To achieve this, we develop a script that works as follows.

1. Run the `git checkout -f ${hash}` to navigate to the desired commit.

2. Run the `git diff --name-only HEAD~1` command to retrieve the files changed between the checked-out commit and the immediate commit before it.
3. Finally, we count the number of files changed, which is the File Churn.

Then, we obtain the **Mean**, **Median**, and **Trimmed Mean**. Table 6 summarizes the results.

Table 6. File Churn metric

Project	Mean	Median	Trimmed Mean	# VFCs
Chromium	5.42	2.00	3.33	1518
Linux	1.88	1.00	1.20	927
ImageMagick	1.38	1.00	1.14	189
FFmpeg	1.19	1.00	1.00	84

Likewise, for the Code Churn metric, Chromium also has higher values than the other projects. For instance, its **Mean** is 5.42 altered files per VFC, which is at least three times higher than the other projects. Regarding Linux, ImageMagick, and FFmpeg, we observe that the **Mean** lies between 1 and 2, which means most VFCs have only one or two files changed per patch.

The Chromium project also presents higher values for **Median** and **Trimmed Mean** than the other projects. In contrast, the **Median** for Linux, ImageMagick, and FFmpeg is 1.0 while the **Trimmed Mean** is close to 1.00. Thus, we notice that in three out of the four selected projects, it is common for fixing patches to alter only one file per VFC.

Therefore, we answer **RQ3** by stating that, in most cases, developers implement vulnerability-fixing patches in approximately one file. Nevertheless, VFCs might involve more than one file in a few cases, as happens for Chromium.

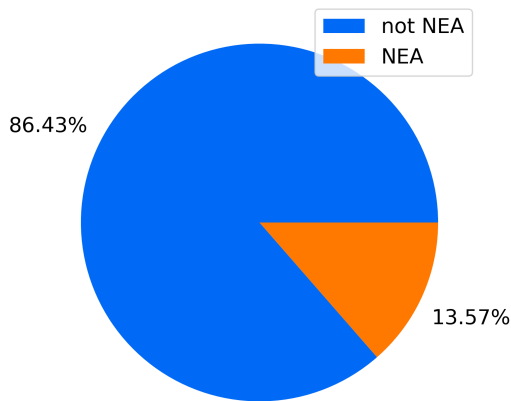
4.4 Authors' Previous Contribution for Altered Code

As explained in Section 3.1, we use the NEA metric to answer **RQ4**, that is, to determine whether the authors of the VFCs are effectively new within the code they alter.

To calculate the NEA metric, we automatically execute *git checkout* to navigate through the VFCs of the four selected projects. Therefore, for each VFC, we identify the altered files using *git diff*. Next, we retrieve the author of the fix and the commit prior to the fix using *git log*. This enables us to navigate to the previous commit once again using *git checkout*. Finally, we execute *git blame* to determine who are the authors that have committed to the target file. We then respond to the metric with "not NEA" if the author of the VFC is found among the authors of the files in the commit prior to the VFC and "NEA" if the author is not found.

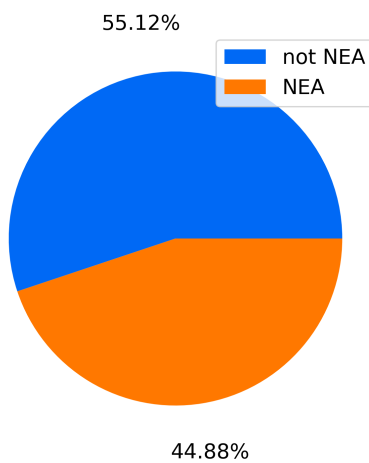
Figure 1 illustrates the results of the NEA metric for Chromium. The percentage of new effective authors in Chromium's VFCs is 13.57%. Therefore, the remaining 86.43% of the authors are effectively new in regard to the code they fix. This result indicates that it is more common for developers to make vulnerability fixes that have contributed to the code they are correcting.

Figure 1. NEA Metric - Chromium



Additionally, Figure 2 displays the results of the NEA metric regarding the Linux project. It is interesting to note that Linux presents the highest percentage of new effective developers making vulnerability fixes (44.88%). Nevertheless, this value does not surpass the number of non-NEA authors working on these fixes. A possible reason for this scenario is that the contribution policies of Linux are less strict than, for example, the Chromium project. This may attract more new developers to contribute to the project.

Figure 2. NEA Metric - Linux



In Figure 3, we present the NEA metric results for the ImageMagick project. For this project, the NEA metric has a lower rate of “NEA” (7.41%). Therefore, ImageMagick receives fewer contributions from new authors. Possible reasons for these differences are: (i) few different developers contribute to the project, (ii) this project is not as popular as Linux or Chromium in terms of the number of commits and contributors, and (iii) only a few contributions to the project end up being approved during the review stage. However, more research is needed to confirm these reasons.

Finally, we show the NEA metric chart for FFmpeg in Figure 4. The occurrence of new effective authors that contribute with VFCs for this project is 23.81%. Therefore, the other 76.19% of VFCs are made by developers who had contributed to the code before.

Thus, we answer **RQ4** by concluding that the author has indeed contributed to the code before the VFCs. However,

Figure 3. NEA Metric - ImageMagick

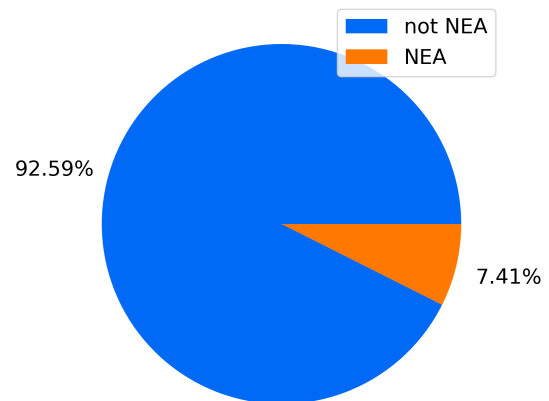
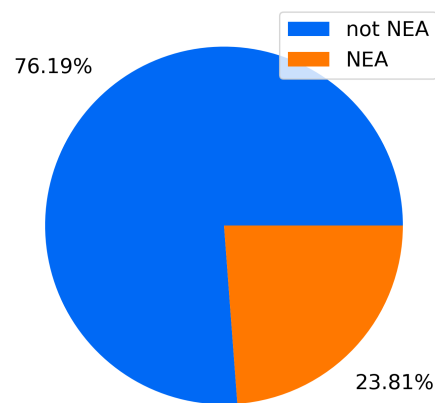


Figure 4. NEA Metric - FFmpeg



it is worth highlighting that the rate of new effective authors might vary among projects. Nevertheless, stating that an author is NEA does not necessarily mean they lack experience in the project. Since they may have contributed to other parts of the project. To address this, we also consider **RQ7** in Section 4.7.

4.5 Correlation Between VCC and VFC Authors

In **RQ5**, we aim to investigate whether the authors of a given VCC are the same authors for the corresponding VFC. In other words, we want to determine whether the authors are fixing vulnerabilities that they previously introduced into the project. To achieve this answer, we calculate the Percentage of Self Churn (PSC) metric as follows:

1. Run the `git diff --name-only HEAD~1` command to obtain the names of the files changed in a given VFC.
2. Run the `git log -1 -p -U0 -- "filename"` with the help of `grep` for each file name obtained in Step 1 to obtain the number of removed or altered lines.
3. Run the `git blame -e "${filename}"` along with the `grep` command for each file obtained in Step 1 to obtain the author of each removed or altered line obtained in Step 2.

4. We count the number of removed or altered lines that were previously written by the author of the given VFC.
5. Lastly, we obtain the percentage of lines removed or altered that were written by the author of the given VFC by the total number of lines removed or altered, which represents the resulting PSC.

We organize the results of the PSC metric in a single scatter plot with the results of the selected projects, as shown in Figure 5. The vertical axis of the plot represents the percentage of affected lines that were previously written by the authors who also created the VFC. The size of the bubbles indicates the frequency of the percentages displayed on the vertical axis. We display the number of occurrences in the largest bubbles for better data visualization. The horizontal axis and the bubble colors indicate the project to which they belong.

It is essential to note that we include bubbles on the vertical axis, representing the occurrences where PSC is undefined, meaning that there are only added lines. Furthermore, we add bubbles representing the total occurrences for each project.

First, we observe the total number of VFCs analyzed for each project. The Chromium project has 1518 commits and Linux has 927, whereas FFmpeg has 84 and ImageMagick has 189 VFCs. Considering that we obtained all the commits from each project in the Big-Vul dataset, the Chromium and the Linux projects have more commits than FFmpeg and ImageMagick. Due to this difference, we consider the proportions of the PSC instead of the total number.

Then, we examine the percentage of commits where the PSC is Undefined for each project analyzed. Undefined PSC represents cases where the commit has only added lines, which means that there are zero altered or removed lines. Therefore, we cannot divide the number of added lines by zero to measure the PSC. Note that the Chromium project has 241 undefined PSC commits, Linux has 221, FFmpeg has 22, and ImageMagick has 42. Even though the total number of Undefined PSC is different, the proportions relative to the total number of commits analyzed are similar. Around 22% to 27% of the commits had only added lines in Linux, FFmpeg, and ImageMagick. Specifically, Chromium is the only project with about 15% of the commits with only added lines.

Next, we analyze the number of VFCs where 100% of the lines changed were previously written by the author of the VFC. In Chromium, that happened in 217 commits, in Linux 49 commits, in FFmpeg 7, and ImageMagick 8. In this way, Chromium has the highest number of VFCs (14.3%) where the PSC metric is equal to 100%, which means that 14.3% of the Chromium VFCs analyzed had all changed lines in the VFC previously written by the same person who authored the VFC. The other three projects have around 4% to 9%.

Finally, we observe the number of VFCs in which all the changed lines were previously written by another author. Chromium has 745 commits, Linux has 579 commits, FFmpeg has 40, and ImageMagick has 97. In this context, Linux has the highest percentage of commits (62.46%) where the code was previously written by someone else.

Despite variations in sample sizes across projects, it is noticeable that the PSC proportion is similar. Therefore, we

Table 7. NAA metric

Project	Mean	Median	Trimmed Mean	# VFCs
Chromium	2.33	1.00	1.43	1518
Linux	1.65	1.00	1.15	927
ImageMagick	1.20	1.00	1.13	189
FFmpeg	1.15	1.00	0.89	84

answer **RQ5** stating that, in most cases, authors are addressing vulnerabilities that they did not contribute to introducing or that do not remove or alter existing code.

Although these results may initially seem contradictory to those of **RQ4**, they actually complement each other. The NEA suggests that the author of the correction had written some of the lines in the code they corrected, whereas the PSC confirms that in certain cases, none of the lines the author modified were their own. Instead, these lines were written by another author.

4.6 Affected Authors

Regarding **RQ6**, we aim to understand if the changes in VFCs affect code written by other developers. Specifically, we want to determine, among the lines altered in the VFC, how many authors had lines affected by the fix other than the VFC author. To answer this question, we use the NAA (Number of Authors Affected) metric [Meneely *et al.*, 2013].

To compute the NAA metric, we follow a set of steps similar to those used to calculate the PSC metric in Section 4.5:

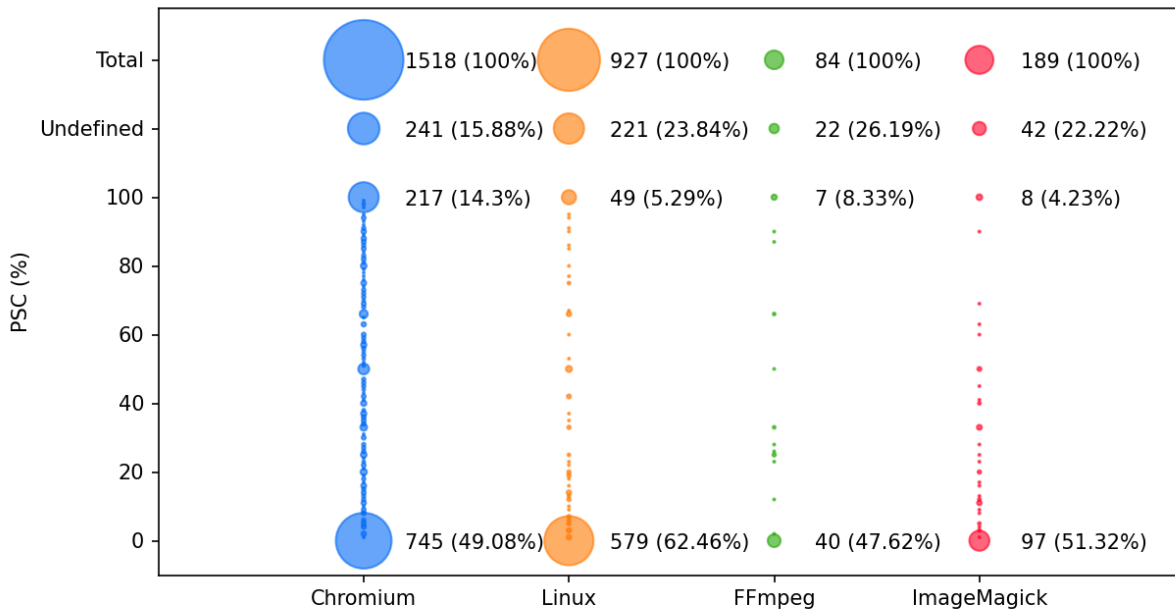
1. Run the `git diff --name-only HEAD~1` command to obtain the names of the files changed in that VFC.
2. Run the `git log -1 -p -U0 -- "filename"` with the help of `grep` for each file name obtained in Step 1 to obtain the number of removed or altered lines.
3. Run the `git blame -e "${filename}"` along with the `grep` command for each file obtained in Step 1 to obtain the author of each removed or altered line obtained in Step 2.
4. We remove duplicate authors from the results of Step 3.
5. Finally, we count the number of authors from the results of Step 4, that is, the Number of Affect Authors.

After obtaining the results of the NAA metric, we calculate the **Mean**, **Median**, and **Trimmed Mean** of the collected data from the above steps. Thus, we present them in Table 7, along with the total number of VFCs.

In the Chromium project, we observe that the **Mean** is 2.33, which may indicate that two authors are affected by VFCs. However, the **Median** is equal to 1.00 and the **Trimmed Mean** is 1.43, indicating the presence of VFC outliers. The same pattern repeats for the other three projects. The **Mean** is higher than the **Median** and **Trimmed Mean** due to outlier VFCs.

Due to the similarity in the values obtained for the **Mean**, **Median**, and **Trimmed Mean** across the different projects under study, we address **RQ6** by stating that it is common for only one author to be affected by VFC. This means that the fix is typically made on code that was authored by a single developer. Furthermore, it should be noted that in the FFmpeg project, the **Trimmed Mean** is less than 1.00. This

Figure 5. PSC Metric



indicates that some VFCs are implemented by the same authors who introduced the VCC. This answer helps to validate the results of the PSC metric for **RQ5** (Section 4.5). Since when NAA is zero, PSC is undefined, and when NAA is between zero and one, PSC is high in most cases, or when NAA is higher than one, PSC is low in most cases. Ultimately, our results from the NAA metric align with the results obtained from the PSC metric.

4.7 Author's Experience for the project

Regarding **RQ7**, we aim to understand the experience of the VFC authors within the project. To answer this question, we calculate the Number of Commits Before (NCB) metric for the authors of each VFC in each project. In other words, we determine the number of commits made by the VFC author before the VFC itself.

To calculate the NCB metric, we use the GitHub REST API [GitHub REST API, 2021] since it allows querying the author's commits in a project. Thus, we iterate through the VFCs of each project, retrieving their authors. Next, we filter and count only the commits made before the VFC. In this way, Table 8 displays the **Mean**, **Median** and **Trimmed Mean** of the NBC results for each project. We also provide the total number of VFCs and the total number of commits in each project's repository as of writing this paper.

For the Chromium project, the **Mean** for NCB is 600.56 commits. However, the **Median** rate is 200.00, which means that there are a few authors who are more experienced than the majority. Furthermore, the **Trimmed Mean** is 290.95, which is closer to the **Median** in case we compare to the **Mean**. This also happens due to a few more experienced authors.

Regarding Linux, the **Mean** for VFC authors is 872.60 commits while the **Median** is 91.00. Note that the former is almost 10 times higher than the latter. Similarly to the Chromium project, a few authors are more experienced than the majority, which contributes to increasing the **Median**. Furthermore, the **Trimmed Mean** is 292.12 because some experienced authors are still included in the sample.

In contrast to Chromium and Linux, the **Mean**, the **Median**, and the **Trimmed Mean** of the VFC authors for ImageMagick are higher: 1,662.77, 1,030.00, and 1,290.59, respectively. These numbers indicate that the VFC authors have long experience in the ImageMagick project. It is also important to note that a few authors committed the 189 VFCs for this project, and they have a large number of total commits.

Surprisingly, our results for the **Mean**, the **Median**, and the **Trimmed Mean** of the FFmpeg project are significantly higher than the other three projects. In this context, only 84 VFCs are present in Big-Vul, which could lead to fewer authors. In fact, we observe that an author committed the majority of these 84 VFCs. Moreover, this author has approximately 30,000 commits⁴. Therefore, this author alone contributes to increasing the FFmpeg results.

Therefore, we answer **RQ7** stating that the authors of VFCs are indeed experienced for the projects that they contribute to. However, the experience in terms of commits prior to VFCs might vary significantly. As shown in Table 8, the authors' experience tends to increase for smaller projects that contain fewer VFCs. Last but not least, in Section 4.8, we discuss our results.

⁴<https://github.com/FFmpeg/FFmpeg/graphs/contributors>

Table 8. NCB Metric

Project	Mean	Median	Trimmed Mean	# VFCs	Total # Commits
Chromium	600.56	200.00	290.95	1918	1,368,120
Linux	872.60	91.00	292.12	927	1,245,577
ImageMagick	1,662.77	1,030.00	1,290.59	189	21,918
FFmpeg	16,473.46	25,662.50	17,208.34	84	113,323

4.8 Discussion

In this section, we discuss the answers to each research question, relate the findings, and explain their implications.

The answers to our research questions may help developers understand which vulnerabilities have the highest number of patches to fix in the studied projects, how the vulnerabilities are fixed, and how experienced the authors who fix the vulnerabilities are, and their relationship with the code they fixed.

These findings can also help researchers create tools that help developers in vulnerability remediation activities. Moreover, they can help project managers to leverage whether developers responsible for the introduction of a vulnerability should be involved in the remediation strategy [Forootani *et al.*, 2022]. Our research questions **RQ4** and **RQ7** examine the experience of the VFC authors; the first concludes that the authors are not effectively new to the code they changed, while the second concludes that the authors are experienced in the project. Thus, the answers to **RQ4** and **RQ7** complement each other to help developers understand the experience of the VFC authors.

Moreover, the results from **RQ5** demonstrate the relationship between the authors of VFC and the code they modified, providing information on their involvement in the actual vulnerability and helping in the decision of who should address it. The results from **RQ6** indicate how many developers contributed to the vulnerable code when it was not authored by the author of the VFC which, more importantly, supports the results obtained from PSC.

For instance, the commit `8ac035c` in the Chromium project reveals that the author of the commit is not a NEA. Also, as of the time of the commit, the author had 100 commits in the project. Moreover, by the PSC metric, 100% of the changes of the commit were made in lines written by the author. Finally, NAA being equal to zero confirms the evidence of the PSC metric. Thus, we conclude that the author is experienced. So, in the context of the discovery of a vulnerability associated with a commit, the project manager could select the author of the vulnerable commit to fix the vulnerability, and the number of commits before that commit may reinforce that choice.

Also, these findings can help project managers and developers to understand the complexity of a fix patch in the number of lines and files to be changed. The **RQ2** and **RQ3** examine the number of lines and files changed in VFCs patches and conclude that not many lines or files changes are required to commit a vulnerability-fixing patch. For example, the same analysis could be done to a software project using our scripts or an automated tool based on our scripts. Then, based on the results, project managers could estimate the mean time to commit a patch to fix a vulnerability in their project once

the problem is discovered.

Finally, we base our study on a vulnerability dataset from real open source projects, the Big-Vul dataset [Fan *et al.*, 2020], which is a curated dataset from the Common Vulnerabilities and Exposures (CVE) [The MITRE Corporation, 2023], and linked to the CWE IDs [The MITRE Corporation, 2024]. Both are used in several other studies such as [Fan *et al.*, 2020; Meneely *et al.*, 2014; Bhandari *et al.*, 2021; Ponta *et al.*, 2019; Perl *et al.*, 2015; Chen *et al.*, 2023] to explore vulnerabilities and fixes characteristics.

Next, we present the threats to validity concerning our study.

4.9 Threats to Validity

In this section, we present the threats to the validity of our study. By following [Wohlin *et al.*, 2012], we organize the threats as Construct validity (Section 4.9.1), External validity (Section 4.9.2), Internal validity (Section 4.9.3), and Conclusion validity (Section 4.9.4).

4.9.1 Construct validity

We select only one metric to answer each of our seven research questions. This might limit the answers to our research questions, since the metrics evaluated may be limited by the project and context studied, and other aspects of vulnerability-fixing could also be investigated to sum with our results. However, to mitigate this threat, we studied seven metrics in four open source projects, and our results comply with related work [Forootani *et al.*, 2022; Piantadosi *et al.*, 2019].

We also plan to use more metrics to reinforce our results, such as, 30-Day NAA by Meneely *et al.* [2013], which can be used to compare with the results of the NAA metric, 30-Day PSC, based on 30-Day-PIC by Meneely *et al.* [2013], to compare with the results of the PSC metric, and churn metrics, such as Relative Churn and 30-Day Churn, to be compared with the results of the Code Churn metric. Since calculating these metrics would require the development of new and more complex scripts compared to those used in this work, we chose not to include them in this study. Instead, we plan to address them in future work.

4.9.2 External validity

Regarding external validity, this research focuses solely on four open-source projects. Consequently, our findings cannot be generalized to all software development projects. Furthermore, our study is based on a single dataset, the Big-Vul dataset, limiting our findings to the commits available within it, which may not fully represent the evaluated projects.

Additionally, exploring other datasets or scraping data for project vulnerabilities and fixes could yield different and new insights. Both tasks are complex since other approaches would need to be developed to explore other datasets or scrape data for other projects. We could, instead, explore other projects in the Big-Vul dataset; however, other projects have a small sample size, which means that exploring those projects would affect the validity of our work. Therefore, future research could investigate additional VFCs beyond the ones we examined. However, this does not diminish the significance of our work, as our findings can assist developers in certain contexts and our methodology can be applied to study other projects and datasets.

Moreover, this research is limited to projects from the Big-Vul dataset, which includes only C/C++ open-source projects. This limitation poses a threat to the validity of our study as we cannot generalize the results to projects written in other programming languages. Additionally, the Big-Vul dataset contains a disproportionate number of vulnerability-fixing commits (VFCs) across projects, leading to variations in the sample sizes analyzed. These differences may influence the results obtained for each metric studied. To address these issues in future work, we plan to include projects from various programming languages and ensure larger, more balanced sample sizes.

4.9.3 Internal validity

As threats to internal validity, our research questions **RQ2**, **RQ3** and **RQ5** answers might be limited by the Code Churn, File Churn, and PSC metrics, since developers might write more lines and change more files than what is actually needed to fix a commit [Herzig and Zeller, 2013] or a single patch may not be sufficient to fix a vulnerability. Regarding **RQ7**, the results might be limited, since some developers prefer to make commits more frequently than others [Rahman and Devanbu, 2011]. As of **RQ4**, the NEA metric results only indicate whether an author has contributed or not to the code they fixed. However, to mitigate this issue, we use **RQ5** and **RQ7** results as complements to **RQ4** results. Also, NEA and NCB metrics could be calculated for all other contributors to the projects to compare with **RQ4**, **RQ5** and **RQ7** results.

4.9.4 Conclusion validity

The Big-Vul dataset is curated from the Common Vulnerabilities and Exposures (CVE), and linked to the corresponding Common Weakness Enumeration (CWE) IDs. Both are used in several other studies to explore vulnerabilities and fix characteristics [Fan *et al.*, 2020; Meneely *et al.*, 2014; Bhandari *et al.*, 2021; Ponta *et al.*, 2019; Perl *et al.*, 2015; Chen *et al.*, 2023]. However, the Big-Vul dataset lacks 306 links from CVE to CWE IDs out of a total of 4,432 VFCs [Fan *et al.*, 2020]. In fact, we can also observe such cases in our answer to **RQ1** regarding the highest number of patches to fix each CVE: the first three VFCs lack their CWE IDs (Table 4). Thus, our work could not draw conclusions about CWEs in the same manner as it could for CVEs. However, in future work, we could circumvent this limitation by finding the missing CWE IDs ourselves. This task would require

replicating the work of [Fan *et al.*, 2020] or developing our own dataset of VFCs.

5 Related Work

In this section, we detail some works related to our study, describing and comparing each to our work.

Forootani *et al.* [2022] investigated the diffusion of vulnerabilities fixed by the author of the vulnerability in software projects, the types of vulnerabilities that are more prone to being fixed by the author, and the time required to fix them. Their results show that 20.55% of the vulnerabilities are fixed by the same author of the vulnerability. In our study, we conducted a similar analysis with the PSC metric, which determines the number of vulnerable lines that were previously written by the author of the fix. We also have similar results to Forootani *et al.* [2022]. Therefore, our study validates their work using different metrics. However, while Forootani *et al.* [2022] focuses only on the authors who fixed the vulnerability they introduced. Our analysis focuses on the experience and collaboration of the authors who fix commits and the characteristics of the fixes.

Piantadosi *et al.* [2019] studied the process of fixing vulnerabilities in open-source projects. The objective of their study is to define the authors who fix vulnerabilities, how long it takes to fix them, and what the procedure is that they follow to fix the vulnerabilities. Their results indicate that the authors of fixes are more experienced than the authors of other commits and that the fixes used to take more than one commit. Piantadosi *et al.* [2019] focuses on comparing the characteristics of the Vulnerability-Fixing Commits with other commits to obtain their results. We analyze the number of lines and files required to fix vulnerabilities, so that our results can lead developers to a new understanding of VFCs. We also provide other information on the authors' experience and collaboration with the code and the project.

In Meneely and Williams [2012], the researchers introduce variations for the socio-technical metric Code Churn. They call them Interactive Churn and Self Churn. Later, they conducted a study of the metrics in the PHP programming language and found that there is a relation between the metrics and the introduction of vulnerabilities. Though we use some of their metrics, in our study, we analyze Vulnerability-Fixing Commits from a dataset to describe the profiles of the commits and their authors; thus, we bring a new approach to understanding the vulnerabilities, which is by studying their fixes. This may help developers understand the vulnerability, the fixes, and their authors, which are more insights to help them tackle the vulnerabilities.

In another study, Meneely *et al.* [2013] used the metrics defined in their later study and defined a new metric, the NEA metric. Then, they answered their research questions using more than one metric for each, trying to characterize the vulnerabilities introduced in the httpd project. In our study, we study more projects: Chromium, Linux, ImageMagick, and FFmpeg, though we used only one metric per research question. We also study vulnerability-fixing commits instead. So, we provide new information to help developers fix vulnerabilities by studying vulnerability fixes. Also, we comple-

ment the work of Meneely *et al.* [2013] by studying more projects with their metrics and providing new metrics.

Bosu *et al.* [2014] studied the characteristics of security vulnerabilities of ten open source software projects. They find that the majority of the contributors are experienced, whereas the less experienced introduce fewer vulnerabilities. In addition, contributors paid to work on the project are more prone to introduce vulnerabilities. In our study, we study vulnerability-fixing commits and their authors' experience. Once again, our results provide an understanding of vulnerabilities by analyzing the characteristics of VFCs, thereby complementing previous work Bosu *et al.* [2014].

In another study, Liu *et al.* [2020] selected five open source software projects and collected data from public vulnerability data sources, including the National Vulnerability Database (NVD) [National Institute of Standards and Technology. National Vulnerability Database (NVD), 2021], Bugzilla reports [The Bugzilla Guide, 2024], security bulletins, and GitHub commits. Then, they present 12 insights on the data collected in three aspects: distribution, dependency, and recurrence. They utilize manual effort and automated effort to collect the data. On the other hand, we utilize only an automated effort. The results of our study offer a perspective that is not addressed by them, and we also provide the scripts to replicate our analysis.

In their study, Perl *et al.* [2015] propose a method for finding potentially dangerous code in code repositories with a significantly lower false-positive rate than comparable systems. They combined code-metric analysis with metadata gathered from code repositories to help code review teams prioritize their work. Their contribution encompasses a CSV dataset, a tool to flag suspicious commits, and a quantitative and qualitative analysis of their approach. Unlike us, they studied with the aim of finding vulnerability code commits, whereas we studied Vulnerability-Fixing Commits. So, their results can help developers and teams find new vulnerable commits in their projects and provide a dataset so that researchers can study those vulnerabilities. We studied VFCs to help developers fix vulnerabilities and provided our scripts to help researchers replicate our study and obtain new findings.

Fan *et al.* [2020] crawled vulnerabilities entries in the CVE database, then they selected CVE entries that have reference links to publicly available Git repositories, where they collected the commits' hashes, code changes, code fixes, etc. Finally, they created a dataset containing the collected data, named Big-Vul, and made it publicly available. Also, they provided some analysis of the dataset, such as the number of vulnerabilities per project, descriptive statistics of the dataset, etc. Unlike them, we did not create a dataset, but we used the Big-Vul dataset as our case study. Thus, we developed seven research questions regarding the vulnerability-fixing commits found in the Big-Vul dataset, then we used seven metrics to answer the research questions and obtain findings not explored in their study.

Gkortzis *et al.* [2018] constructed a dataset that correlates software metrics from open source projects with their security vulnerabilities, made the dataset publicly available, and provided a guide on how to use it for future research while demonstrating some findings of the dataset. Their work is

different from ours because they developed a vulnerability dataset, whereas we study one. Furthermore, while they present an analysis of their dataset, we calculated seven different metrics on VFCs. However, their dataset has potential for future research to replicate our study.

In another work, Ponta *et al.* [2019] manually curated a dataset of open source software vulnerabilities and commits to fixing them. They also offer an analysis of the dataset, including details such as the number of vulnerabilities per year, the number of commits, and more. Their study could also be used to replicate our work, since they provide commit fixes to facilitate analyzing fixes. In addition, Bhandari *et al.* [2021] surveyed existing security vulnerability-related datasets to discuss their strengths and weaknesses. Then, they propose a method to automatically collect and curate a comprehensive vulnerability dataset from CVE records in the NVD. They implemented the proposed method and made it publicly available. Unlike ours, their work explores the creation of new vulnerability datasets, whereas our study examines existing datasets to draw new conclusions and uncover findings that assist developers in characterizing vulnerability-fixing commits.

6 Conclusion

In this work, we present an analysis of vulnerability-fixing commits and their authors in four open-source software projects, Chromium, Linux, ImageMagick, and Ffmpeg. We gathered data from the Big-Vul dataset, the GitHub REST API, and the actual projects' repositories to answer the seven research questions we defined. Our research questions aim to investigate the vulnerabilities, the commits that fix them, and their authors. We applied seven metrics (NPS, Code Churn, File Churn, NAA, NEA, PSC, and NBC) to answer our research questions.

The purpose of this study was to investigate the characteristics of Vulnerability-Fixing Commits, which offer an understanding of the vulnerabilities, their fixes, and their authors. Thus, developers could use our findings to understand the complexity of a VFC in lines of code or the number of files. Also, they may help project managers assign developers to fix vulnerabilities based on developers' experience in the code and in the project, or even in the vulnerability itself.

The results of this work indicate that vulnerabilities with the highest number of patches to solve found in the Big-Vul dataset are highly specific to the projects where they occur. We found that most of the fixes are implemented by experienced developers in the project. Additionally, we observed that Vulnerability-Fixing Commits typically involve minimal changes to files and lines. Moreover, the authors of the VFCs are familiar with the code, as they have made prior contributions to it. However, the specific lines they modify are often authored by other contributors.

We believe that our findings could improve the productivity of developers by offering insights into the characteristics of VFCs. For example, in the case of the Chromium project, it would be more effective to assign an experienced developer to resolve a vulnerability, as 86.43% of the VFCs in this project are authored by experienced developers (Fig-

ure 1). Therefore, the Chromium team could potentially reduce the time required to produce VFCs by ensuring that less experienced developers are not assigned to these tasks. Furthermore, since our findings indicate that vulnerabilities are often fixed by developers who did not introduce them (Figure 5), teams can avoid unnecessary delays by not waiting for the original authors of the vulnerable code to address the issue.

Additionally, better understanding of VFCs could aid teams in assigning responsibility for fixing vulnerabilities based on developers' experience with the code and project, as well as their involvement in the vulnerability's introduction. Furthermore, we have made all scripts and data publicly accessible to facilitate the replication of the study and support future research.

In future work, our aim is to extend this research by developing automated tools to help project managers and developers better understand the complexity of the fixes and help assign the most suitable authors to fix vulnerabilities. In addition, we plan to improve our analysis by incorporating new research questions and additional metrics to strengthen our findings. Furthermore, we intend to replicate this study with additional datasets. Another interesting approach is that we could experiment with new metrics that focus on areas such as team collaboration dynamics or the influence of different types of code changes on the effectiveness of fixes [den Besten *et al.*, 2021; Spagnoletti *et al.*, 2022]. Moreover, we could improve the experimental design by incorporating machine learning models [Hanif *et al.*, 2021; Ghaffarian and Shahriari, 2017; Harer *et al.*, 2018]. This approach would enable us to uncover nuanced patterns that extend beyond the capabilities of current metrics. Finally, we could also investigate the experience of the VFC author using an approach such as the truck factor [Avelino *et al.*, 2016; Rigby *et al.*, 2016; Cosentino *et al.*, 2015]. Thus, we would be able to conclude whether the VFC authors have a long experience for the project that they contribute to.

Declarations

Funding

This study was partially financed by the Fundação de Amparo a Ciência e Tecnologia do Estado de Pernambuco (FACEPE) under grant number IBPG-0475-1.03/23.

Authors' Contributions

Rodrigo Andrade contributed to the conception of this study. Rodrigo Andrade contributed to the research questions. Vinícius Almeida contributed to the search for vulnerability datasets, the selection of open-source projects, and the development of the scripts to obtain the metrics results. Rodrigo Andrade and Vinícius Almeida are the main contributors and writers of this manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they do not have competing interests.

Availability of data and materials

The scripts developed and used in this study and the resulting data are available in our online Appendix Almeida and Andrade [2024].

References

- Allen, J., Barnum, S., Ellison, R., McGraw, G., and Mead, N. (2006). *Software Security Engineering*. Addison-Wesley Professional. Book.
- Almeida, V. and Andrade, R. (2024). Online appendix. Available at: <<https://github.com/vinesnts/investigando-conjuntos-de-dados-de-vulnerabilidades-scripts-e-dados>>.
- Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2016). A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. DOI: 10.1109/ICPC.2016.7503718.
- Basili, V., Caldiera, G., and Rombach, D. H. (1994). The goal question metric approach. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering*, pages 528–532. Wiley, New Jersey. Available at: <https://www.ecs.csun.edu/~rllingard/COMP587/gqm.pdf>.
- Bhandari, G., Naseer, A., and Moonen, L. (2021). Cve-fixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 30–39. DOI: 10.5281/zenodo.5111494.
- Bosu, A., Carver, J. C., Hafiz, M., Hilley, P., and Janni, D. (2014). Identifying the characteristics of vulnerable code changes: An empirical study. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014*, page 257–268, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2635868.2635880.
- Chen, Y., Ding, Z., Alowain, L., Chen, X., and Wagner, D. (2023). Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection. *RAID '23*, page 654–668, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3607199.3607242.
- Cosentino, V., Izquierdo, J. L. C., and Cabot, J. (2015). Assessing the bus factor of git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 499–503. DOI: 10.1109/SANER.2015.7081864.
- CVE-2011-3110 (2023). CVE-2011-3110. Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3110>>.
- CVE-2012-2827 (2023). CVE-2012-2827. Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2827>>.
- CVE-2013-0892 (2023). CVE-2013-0892. Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-0892>>.
- CVE-2015-1265 (2023). CVE-2015-1265. Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1265>>.

- CVE-2017-6903 (2023). CVE-2017-6903. Available at: <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-6903>>.
- den Besten, M., Amrit, C., Capiluppi, A., and Robles, G. (2021). Collaboration and innovation dynamics in software ecosystems: A technology management research perspective. *IEEE Transactions on Engineering Management*, 68(5):1532–1537. DOI: 10.1109/TEM.2020.3015969.
- Fan, J., Li, Y., Wang, S., and Nguyen, T. N. (2020). A c/c++ code vulnerability dataset with code changes and cve summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20*, page 508–512, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3379597.3387501.
- Ffmpeg (2023). Ffmpeg. Available at: <<https://ffmpeg.org/about.html>>.
- Forootani, S., Sorbo, A. D., and Visaggio, C. A. (2022). An exploratory study on self-fixed software vulnerabilities in oss projects. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 90–100. DOI: 10.1109/SANER53432.2022.00023.
- Ghaffarian, S. M. and Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. 50(4). DOI: 10.1145/3092566.
- GitHub REST API (2021). Available at: <<https://docs.github.com/pt/rest>>.
- Gkortzis, A., Mitropoulos, D., and Spinellis, D. (2018). Vulnoss: A dataset of security vulnerabilities in open-source systems. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR '18*, page 18–21, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3196398.3196454.
- Graf, J., Hecker, M., and Mohr, M. (2013). Using joana for information flow control in java programs - a practical guide. In Wagner, S. and Lichter, H., editors, *Software Engineering 2013 - Workshopband*, pages 123–138, Bonn. Gesellschaft für Informatik e.V. Available at: <https://dl.gi.de/items/de5f4132-ce46-4f03-b578-25bd78b65928>.
- Hanif, H., Md Nasir, M. H. N., Ab Razak, M. F., Firdaus, A., and Anuar, N. B. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications*, 179:103009. DOI: 10.1016/j.jnca.2021.103000.
- Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., Hamilton, L. H., Centeno, G. I., Key, J. R., Ellingwood, P. M., Antelman, E., Mackay, A., McConley, M. W., Opper, J. M., Chin, P., and Lazovich, T. (2018). Automated software vulnerability detection with machine learning.
- Herzig, K. and Zeller, A. (2013). The impact of tangled code changes. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 121–130. DOI: 10.1109/MSR.2013.6624018.
- ImageMagick (2023). Imagemagick. Available at: <<https://imagemagick.org/index.php>>.
- Krsul, I. V. (1998). Software vulnerability analysis. Available at: <https://www.proquest.com/docview/304449527?fromopenview=true&pq-origsite=gscholar&sourcetype=Dissertations%20%20Theses>.
- Liu, B., Meng, G., Zou, W., Gong, Q., Li, F., Lin, M., Sun, D., Huo, W., and Zhang, C. (2020). A large-scale empirical study on vulnerability distribution within projects and the lessons learned. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20*, page 1547–1559, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3377811.3380923.
- McGraw, G., Allen, J. H., Mead, N., Ellison, R. J., and Barnum, S. (2008). *Software Security Engineering: A Guide for Project Managers*. Pearson Education (US), EUA. Book.
- Meneely, A., Srinivasan, H., Musa, A., Tejada, A. R., Mokary, M., and Spates, B. (2013). When a patch goes bad: Exploring the properties of vulnerability-contributing commits. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 65–74. DOI: 10.1109/ESEM.2013.19.
- Meneely, A., Tejada, A. C. R., Spates, B., Shannon Trudeau, D. N., Whitlock, K., Ketant, C., and Davis, K. (2014). An empirical investigation of socio-technical code review metrics and security vulnerabilities. In *Proceedings of the 6th International Workshop on Social Software Engineering*, pages 37–44. DOI: 10.1145/2661685.2661687.
- Meneely, A. and Williams, O. (2012). Interactive churn metrics: Socio-technical variants of code churn. *SIGSOFT Softw. Eng. Notes*, 37(6):1–6. DOI: 10.1145/2382756.2382785.
- National Institute of Standards and Technology. National Vulnerability Database (NVD) (2021). Available at: <<https://nvd.nist.gov/>>.
- OpenJK (2023). Available at: <<https://github.com/JACoders/OpenJK>>.
- Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., and Acar, Y. (2015). Vcfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15*, page 426–437, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2810103.2813604.
- Piantadosi, V., Scalabrino, S., and Oliveto, R. (2019). Fixing of security vulnerabilities in open source projects: A case study of apache http server and apache tomcat. *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 68–78. DOI: 10.1109/ICST.2019.00017.
- Ponta, S. E., Plate, H., Sabetta, A., Bezzi, M., and Dangremont, C. (2019). A manually-curated dataset of fixes to vulnerabilities of open-source software. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 383–387. DOI: 10.1109/MSR.2019.00064.
- Rahman, F. and Devanbu, P. (2011). Ownership, experi-

- ence and defects: a fine-grained study of authorship. In *International Conference on Software Engineering, ICSE '11*, page 491–500, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/1985793.1985860.
- Rigby, P. C., Zhu, Y. C., Donadelli, S. M., and Mockus, A. (2016). Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya. In *Proceedings of the 38th International Conference on Software Engineering, ICSE '16*, page 1006–1016, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2884781.2884851.
- Spagnoletti, P., Kazemargi, N., and Prencipe, A. (2022). Agile practices and organizational agility in software ecosystems. *IEEE Transactions on Engineering Management*, 69(6):3604–3617. DOI: 10.1109/TEM.2021.3110105.
- The Bugzilla Guide (2024). Available at: <<https://www.bugzilla.org/docs/4.2/en/html/>>.
- The Chromium Projects (2023). Chromium. Available at: <<https://www.chromium.org/chromium-projects/>>.
- The Chromium Repository on GitHub (2023). Github. Available at: <<https://github.com/chromium/chromium>>.
- The FFmpeg Repository on GitHub (2023). Github. Available at: <<https://github.com/FFmpeg/FFmpeg>>.
- The ImageMagick Repository on GitHub (2023). Github. Available at: <<https://github.com/ImageMagick/ImageMagick>>.
- The Linux Foundation (2017). 2017 linux kernel development report. Available at: https://linuxclass.heinz.cmu.edu/doc/LinuxKernelReport_2017.pdf.
- The Linux Kernel Archives (2023). Linux. Available at: <https://www.kernel.org/>.
- The Linux Repository on GitHub (2023). Github. Available at: <https://github.com/torvalds/linux>.
- The MITRE Corporation (2023). Common vulnerabilities and exposures (cve). Available at: <<https://cve.mitre.org>>.
- The MITRE Corporation (2024). Common weakness enumeration. Available at <<https://cwe.mitre.org/>>.
- W3Schools (2023). Available at: <<https://www.w3schools.com/browsers/>>.
- Wan, L. (2019). *Automated vulnerability detection system based on commit messages*. PhD thesis. PhD Thesis.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media. DOI: 10.1007/978-3-662-69306-3.
- Wood, A. and Stankovic, J. (2002). Denial of service in sensor networks. *Computer*, 35(10):54–62. DOI: 10.1109/MC.2002.1039518.