Investigating vulnerability datasets

Rodrigo Andrade
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
rodrigo.andrade@ufape.edu.br

Vinícius Santos
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
v.santos0406@gmail.com

ABSTRACT

Insecure software can cause severe damage to user experience and privacy. Therefore, developers should be able to prevent software vulnerabilities. However, detecting such problems is expensive and time consuming. To mitigate this issue, researchers propose vulnerability datasets to make it easier to investigate its properties. In this work, we investigate one dataset to better understand common vulnerabilities, the authors who introduce them to open-source projects, and commit properties. Thus, we use as case study the Big-Vul dataset to help us answering the six research questions we define for this work. Our preliminary results indicate that the most common vulnerabilities occur in the Chromium project. Furthermore, mostly experienced authors are responsible for introducing these vulnerabilities. Last but not least, we conclude that such findings could help developers on detecting vulnerabilities.

KEYWORDS

Vulnerability; Datasets; Commits; CVE.

1 INTRODUCTION

Security is a major concern for software developers. Insecure software is expensive to maintain and it is harmful for user experience and privacy [3]. Thus, properly dealing with security is important.

To build secure software, developers rely on preventing software vulnerabilities [15]. They are instances of an error in the specification, development, or configuration of software such that its execution can violate a security policy. Software vulnerabilities might have consequences that go beyond the annoyance of common software system failures [12]. The exploitation of vulnerabilities can affect the privacy of users and have disastrous effects.

Unfortunately, finding software vulnerabilities requires expertise in the specific system and in software security [1, 14]. Thus, finding and fixing vulnerabilities are costly and time consuming [19]. To circumvent this problem, researchers propose a number of solutions such as tools to automatically detect certain kinds of vulnerabilities [11, 18] and datasets of disclosed vulnerabilities [7, 10, 19]. In particular, these datasets are a collection of vulnerabilities records usually linked to an existing vulnerability database such as the Common Vulnerabilities and Exposures (CVE) [4]. Thus, they allow developers and researchers to investigate a number of properties related to vulnerabilities.

In this context, our goal is to investigate, initially, the most common vulnerabilities, characteristics of authors who introduced them, properties of their commit messages, and possible relation between vulnerable commits and vulnerability severity. Therefore, the results of our assessment could help developers and researchers to find software vulnerabilities.

To achieve this goal, we select a dataset called Big-Vul [7], which contains more than three thousand vulnerability records across more than three hundred open-source projects. Then, we define six research questions regarding our investigation goals aforementioned. We answer these questions by querying Big-Vul to gather relevant data, such as the number of CVEs occurrences within this dataset. We also calculate metrics related to the number of commits submitted by authors to help answering our research questions.

Our results indicate that the most common CVEs happen for the Chromium project[1]. Indeed, four out of five CVEs are from this project even though other highly supported projects have vulnerability records (e.g., Linux and Git). As a consequence, we define hypotheses to explain this fact, which we aim at validating in future work. Moreover, our results show that experienced developers are more likely to introduce the most common vulnerabilities. Since this is an ongoing work, we should have more interesting results soon. For example, our preliminary data indicate that small size commits tend to introduce the most severe vulnerabilities.

We also provide a discussion on our current results and the benefits of our outcomes. For instance, our findings could help developers to improve their productivity on detecting vulnerabilities by knowing that experienced developers are more prone to introduce issues.

To sum up, our contribution in this work is the following:

- An experience report on investigating the Big-Vul dataset;
- Findings about the most common CVEs in Big-Vul;
- Discussion on properties of authors who introduced vulnerabilities;

The remainder of this work is organized as follows. Section 2 explains concepts for better understanding, such as software vulnerabilities. And also discusses related work. In Section 3, we introduce our research method. In particular, we explain our case study and research questions. Section 4 contains the explanation of our results as well as a discussion about them. At last, Section 5 concludes our study and discusses future work.

2 BACKGROUND & RELATED WORK

GitHub is a platform for source code storage and collaboration among a number of developers. Nowadays, it has more than 70 million users and 220 million repositories [9]. Many of these repositories are public, thus anyone can view the developers' activities. The available activities are mainly, but not limited to, actions around issues, pull requests, and commits. It is also possible to view property of repositories, such as number of stars, forks, and contributors.

The large amount of public data on GitHub makes it possible for researchers to easily mine the repository data [6]. A number of tools and datasets are available to assist researchers in mining such data [2, 8, 21]. Researchers can use such datasets for a number of different purposes. One important goal is investigating vulnerabilities for open-source software [10].

[1] https://github.com/chromium

In particular, a few available datasets assist researchers to mine repositories with the goal to better understand software vulnerabilities [7, 10, 13, 18, 19]. Investigating characteristics of vulnerabilities for open-source software can provide findings that should lead to the development of more secure systems [10].

In this context, in Section 2.1, we explain software vulnerabilities, whereas Section 2.2 discusses datasets for software vulnerabilities. At last, Section 2.3 explains previous work on repository analysis regarding software vulnerability.

2.1 Software vulnerabilities

Software vulnerabilities are instances of errors that violate an implicit or explicit security policy (e.g., a user cannot read someone else's password) [12]. The key to secure software lies when developers are able to prevent such vulnerabilities [15].

In this context, we can classify vulnerabilities by some of their characteristics: availability, integrity, and confidentiality impact, access complexity, type, description, and so on. For instance, Table 1 shows a vulnerability record and some of its details[2].

Table 1: Vulnerability record details

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 MAPHUGETLB mmap operations. 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 MAPHUGETLB mmap operations.
Confidentiality impact	None
Integrity impact	None
Availability impact	Complete
Type	Denial of Service
Access Complexity	Low
Severity (CVSS) [5]	4.9

The description states that a memory leak occurs in a particular file in the Linux kernel for versions prior to 3.4.2. The availability impact is complete because there might be a total shutdown of the system when attackers exploit this Denial of Service vulnerability. Furthermore, there is no impact for integrity and confidentiality.

The example of Table 1 is a software vulnerability because attackers are able to violate an implicit security policy such as *local users cannot execute invalid mmap operations*. Developers fixed this issue by defining a new function to check for invalid mmap operations as showed in Listing 1. The `if` statement in line 4 prevent users from inappropriately provoking a memory leak, as stated in Table 1.

Listing 1: CVE-2012-2390 fix

```
1 static void resv_map_put(struct vm_area_struct *vma)
2 {
3  struct resv_map *reservations = vma_resv_map(vma);
4  if (!reservations)
5   return;
6  kref_put(&reservations->refs, resv_map_release);
7 }
```

Understanding these software vulnerabilities along with their addition and fix during software development is important to build software more secure [10]. One way to achieve this goal is to investigate datasets for software vulnerabilities.

[2]https://www.cvedetails.com/cve/CVE-2012-2390/

2.2 Datasets for software vulnerabilities

Datasets for software vulnerabilities allow researchers to investigate projects for software defects regarding security. The data collection usually starts from an existing vulnerability database, such as the Common Vulnerabilities and Exposures (CVE) [4] and the National Vulnerability Database (NVD) [17]. For example, the CVE program maintains a list of records where each one contains an identification number, a description, and at least one public reference for publicly known vulnerabilities[3]. Any party can feed these databases by updating a vulnerability record they have found in their projects.

From such databases, researchers can crawl their records to gather useful information such as the CVE identification, vulnerability classification, publish date and etc. Then, they can select CVE or NVD records that have reference links of publicly GitHub repositories [7, 13]. These links lead to actual code commits containing the addition of a known vulnerability to a given project stored on GitHub. From these commits, researchers can gather specific and useful information around the vulnerability. For instance, code commit date, author, commit message, files changed, programming language, time frame before and after the fix, and etc.

Moreover, we can use these datasets to better understand how developers added and fixed a given vulnerability within open-source software stored on GitHub. For example, we can determine the most common type of vulnerability for a given project and the reasons why it happened (e.g., inexperienced author altering critical security related files). Developers can benefit from such knowledge to prevent adding new vulnerabilities, which leads to more secure software [10].

2.3 Repository analysis for software vulnerabilities

Meneely et al. conduct an analysis on code repository that relates vulnerabilities to version control commits [15]. In this context, they find 124 commits that introduced vulnerabilities in the Apache HTTP server. Additionally, they analyze characteristics of these commits like their size and author profile. However, these researchers manually analyze the Apache HTTP server instead of using an existing dataset. Thereby, this work approaches only one project. In another related work, Meneely et al. investigate the relation between collaborative code reviews and vulnerabilities [16]. They analyze a number of code reviews and commits of the Chromium browser project to conclude that lack of collaborator familiarity and security experience are risk factors to miss a vulnerability.

Zaman et al. study how security and performance issues differ from other types of problems in the Firefox project [20]. The authors conclude that security-related bugs are fixed faster, involve more developers, and impact more files. They collect bug information from CVS and Bugzilla, which allows a manual classification of bug types and fixes. Unlike our work, they do not investigate vulnerability properties such as integrity, confidentiality, and availability.

Besides collecting a dataset, Liu et al. analyze whether vulnerability occurrence location, time, type, author, and dependency could

[3]https://www.cvedetails.com/cve/CVE-2009-1194/

facilitate its detection [13]. They draw interesting findings such as stating that a higher complexity of code cannot necessarily induce more vulnerabilities. Different from our work, the authors use only their own dataset. Our goal in this work is to use a larger sample of dataset to confirm or reject previous conclusions like those in Liu et al. work.

3 RESEARCH METHOD

In this section, we explain how we conduct our work. In particular, we present our case study in Section 3.1 and we discuss our research questions in Section 3.2.

3.1 Case study: Big-Vul

Currently, we conduct our work using Big-Vul [7]. It is a large dataset containing vulnerability records of open-source GitHub projects written in C/C++. In addition, this dataset also relates the vulnerability occurrence with the corresponding CVE identification [4]. In total, it has 3754 code vulnerabilities from 348 open-source projects encompassing 91 different vulnerability types.

Big-Vul is intended to enable analysis on the characteristics of different vulnerabilities and code changes that introduces them as well as improving the detection and fixing of vulnerabilities [7]. Besides the CVE identification, Big-Vul's record contains details about availability, integrity, confidentiality, publish date, summary, vulnerability classification, commit hash and message that introduced the issue, files changed, language (C or C++), project name, and commit hashes from before and after the fix.

We choose the Big-Vul as the first dataset of our work because it is free of cost and contains scripts to help unpacking and analyzing the data. Additionally, it has more than 3000 vulnerability records, which is a large number comparing to others [13, 18, 19].

To analyze this dataset for our purpose, we define scripts to parse its data to a MySQL database. This way, we could simplify the queries we need to execute in order to gather the necessary data for this work. The resulting database contains 3542 tuples where each one represents a different vulnerability record. Furthermore, these vulnerabilities occurred throughout 322 different projects. As future work, we plan to also conduct our study for additional datasets.

3.2 Research questions

As mentioned in Section 1, our goal is to investigate commits that introduce vulnerabilities. Thus, we propose the following research questions:

- **RQ1**: What are the five mostly introduced CVE regarding the selected dataset?
  RQ1.1: How distributed across different projects these CVEs are?
  RQ1.2: How experienced are the authors who committed the CVEs?
  RQ1.3: What are the main properties of the commits that introduce the CVEs?
- **RQ2**: Can we predict a vulnerable commit based on its commit message?
- **RQ3**: What properties regarding a vulnerable commit has influence on the severity of the introduced vulnerability?

The answer to **RQ1** could help us understand properties related to commits that introduce common CVE. For instance, we could investigate whether the most commons CVEs are specific to one or more projects, their programming languages and severity (*RQ1.1*). Besides that, it is interesting to investigate characteristics of authors whose contributions introduced the CVE, such as how experienced they are (*RQ1.2*). We could express author experience as the number of commits she has submitted to a given project prior to introducing the CVE. Furthermore, investigating commit properties like size (i.e., number of additions and deletions), date, commit messages, or number of altered files could bring insights to help developers detecting vulnerable commits (*RQ1.3*). Thus, our conclusions could improve the detection of such commits. For example, a developer responsible for reviewing and merging someone else's contribution could perform these tasks more carefully for commits presenting certain properties (e.g., large size and few altered files).

Regarding **RQ2**, we are interested in the most common words that occur on vulnerable commits' messages. Therefore, we aim at investigating whether there is a statistical relation that could help developers to detect vulnerabilities. For instance, in case there is a relation between a set of words included in commit messages and vulnerable changes, developers could focus on seeking vulnerabilities on such commits.

Currently, our last research question **RQ3** regards vulnerable commit properties that influence vulnerability severity. In this work, we use the Common Vulnerability Scoring System (CVSS) framework [5] to measure the software vulnerability severity. Each CVE has an assigned CVSS, as illustrated in Table 1. In this context, we plan to investigate whether the number of altered files within a commit has impact on the vulnerability severity. Thus, we could help developers to narrow down the commits they have to prioritize in order to find vulnerabilities. Other interesting commit properties are size, date (e.g., Fridays are more susceptible?), programming language used to implement the changes and kind of altered files (e.g., database accessing, user interface).

4 RESULTS AND DISCUSSION

Currently, we have results regarding **RQ1** and its two first minor research questions *RQ1.1* and *RQ1.2*. Thus, we answer them based only on the Big-Vul case study in Section 4.1. Albeit we propose and discuss *RQ1.3*, **RQ2**, and **RQ3** in this work, we are currently working on their answers. Therefore, we aim at tackling these questions in future work. However, we briefly discuss our preliminary results for *RQ1.3* in Section 4.1. We also discuss our finding in Section 4.2 and some threats to the validity of this work in Section 4.3.

4.1 RQ1: What are the five mostly introduced CVE regarding the selected dataset?

To answer this research question, we query the Big-Vul dataset to check which CVEs present the most occurrences. Table 2 illustrates the top five CVEs in this context. For instance, the `CVE-2012-2875` occurred 15 times for the Chromium project. The resulting vulnerability presents a severity score of 6.8 in the CVSS scale (max is 10). The vulnerability type is undefined and it happens due to issues in the PDF functionality for the Google Chrome web browser. Furthermore, the `CVE-2017-6903` presents the highest CVSS, that is, this

CVE represent the most severe issue among the five CVEs. Indeed, it can cause a complete confidentiality, integrity, and availability impact and attackers do not need to authenticate in order to exploit the resulting vulnerability.

The Big-Vul dataset contains vulnerability information from large and widely supported projects such as Linux, Bitcoin, Git, and MySQL Server. However, surprisingly, four out of the five most common CVE are from the Chromium project[4], which answers *RQ1.1*. This finding brings interesting research opportunities. For example, we could investigate the reason why the software developed within the Chromium project are supposedly more insecure than others. Based on our experience, we consider the hypotheses below:

- H1: Code review process is less rigid than other large projects;
- H2: The application domain (Web Browser) is susceptible for vulnerability introduction;
- H3: The Chromium project is used as case study for more studies;

Although we plan to investigate these hypotheses in future work, we invite other researchers to also reason about the vulnerabilities regarding the Chromium project. For H1, we plan to compare the Chromium code review process to another less vulnerability-prone project. We could check the number of contributions being rejected, the number of asked corrections, and also how experience the developers that are responsible for code reviewing are. Moreover, for H2, we aim at investigating whether the vulnerabilities are browser-related and how does it impact on a higher number of CVE comparing to another project of a different domain. At last, we believe that one possible reason for more CVEs in the Chromium project is due to it being used for many studies regarding software vulnerability. Thus, its flaws are better documented than other large projects. Nonetheless, we need to investigate H3 to propose conclusions in this context.

Regarding *RQ1.2*, we state that most of the authors who introduced the vulnerabilities are experienced. In this work, we determine whether a developer is experienced based on data concerning the number of commits per month prior to introducing the vulnerability identified by the CVE. In future work, we also plan to use the *NEA?* metric [15] to ratify our findings. The *NEA?* metric is defined as a nominal "Yes" or "No" for when an author is a New Effective Author, or has no commit for the files where the vulnerability is introduced prior to the problematic commit.

As showed in Table 3, we measure data regarding the number of commits per month from authors who have introduced the vulnerability identified by each **CVE-ID**. For instance, regarding the CVE-2012-2875, authors have committed an average of 88 commits per month before the introduction of such CVE. Additionally, the median is 19, which means that the authors are equally distributed above and below this threshold. One of the authors submitted only one commit before introducing this CVE while another author submitted 743 commits. Therefore, the former is inexperienced and the latter is experienced.

To sum up, the authors who introduced the vulnerabilities are more likely to be experienced because they have contributed with at least 13 commits per month on average for the projects where the CVE were detected. Despite existing outliers such as the author who committed 743 times, we observe that the lowest median (seven) demonstrates that the majority of authors are not one time contributors.

The answer to *RQ1.3* is an ongoing work. Thus, we plan to investigate whether the commit size in terms of number of additions and deletions impact the proneness to increase or decrease the vulnerability severity. To our knowledge, related work only concern about the proneness of commit size to introduce vulnerability. However, they do not consider the severity fact. Unexpectedly, our preliminary assessment indicates that commits of small sizes tend to increase vulnerability severity considering the CVSS framework [5]. Therefore, we leave the detailed answer to *RQ1.3* for future work.

4.2 Discussion

By answering our research questions, we intend to help developers to detect software vulnerabilities. The goal is to make this task less costly and time consuming.

First, the vulnerability detection could demand less people and resource involved because developers can focus on commits and authors more prone to introduce such issues. Therefore, it could decrease the number of developers assigned to review source code and history. Also, it could decrease the need to use resource-expensive (e.g., memory consumption) tools that analyze source code, such as VCCFinder [18] or JOANA [11].

Second, with the knowledge we aim at providing, developers could be able to focus their effort to detect software vulnerabilities mainly on more susceptible commits. For example, commits submitted by experienced developers. Thus, they could perform this task faster than analyzing all the existing commits equally.

Currently, we only have few results to actually help developers in this context. However, after answering the defined research questions and validating or rejecting the three hypotheses, we believe that our conclusions could decrease the cost and time to find software vulnerabilities.

4.3 Threats to validity

Using only one dataset might affect our ability to draw the correct conclusion. Indeed, analyzing other datasets could lead us to different results. However, we mitigate this issue by selecting a large dataset (Big-Vul) which contains a number of vulnerabilities from several different software projects. Nonetheless, in future work, we plan to also analyze other datasets such as the VulinOSS [10] and [19].

Another important threat regards the fact that Big-Vul presents only vulnerabilities identified in C or C++ software projects. Therefore, its vulnerability records could be specific to these programming languages. In this context, we cannot generalize our results for alternative datasets that consider projects written in other languages like Java. In this way, we could minimize this threat when we analyzing a larger number of datasets in future work.

5 CONCLUSIONS & FUTURE WORK

In this study, we investigate properties related to vulnerabilities, commits, and their authors. To achieve that goal, we studied the Big-Vul dataset, which contains thousands of vulnerabilities entries

[4]https://github.com/chromium

Table 2: Most common CVEs for the Big-Vul dataset

CVE-ID	N° of occur.	Projects	CVSS	Vulnerability type	Description
CVE-2012-2875	15	chromium	6.8	Undefined	Multiple unspecified vulnerabilities in the PDF functionality in Google Chrome before 22.0.1229.79 allow remote attackers to have an unknown impact via a crafted document.
CVE-2015-1265	14	chromium	7.5	Denial of Service	Multiple unspecified vulnerabilities in Google Chrome before 43.0.2357.65 allow attackers to cause a denial of service or possibly have other impact via unknown vectors.
CVE-2013-0892	7	chromium	7.5	Denial of Service	Multiple unspecified vulnerabilities in the IPC layer in Google Chrome before 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 possibly have other impact via unknown vectors.
CVE-2017-6903	7	OpenJK ioq3 iortcw	9.3	Undefined	In ioquake3 before 2017-03-14, the auto-downloading feature has insufficient content restrictions. In OpenJK, iortcw, a malicious auto-downloaded file can trigger loading of crafted auto-downloaded files as native code DLLs. Also, it can contain configuration defaults that override the user, and executable bytecode in it can set configuration variables to values that will result in unwanted native code DLLs being loaded, resulting in sandbox escape.
CVE-2011-3110	6	chromium	7.5	Denial of Service Overflow	The PDF functionality in Google Chrome before 19.0.1084.52 allows remote attackers to cause a denial of service or possibly have unspecified other impact via vectors that trigger out-of-bounds write operations.

Table 3: Authors experience data

CVE-ID	Average	Median	Min	Max
CVE-2012-2875	88	19	1	743
CVE-2015-1265	15	9	1	153
CVE-2013-0892	13	9	1	68
CVE-2017-6903	13	7	1	119
CVE-2011-3110	95	25	1	651

for a number of projects. In particular, we defined three major research questions plus three minor ones. We use them to guide us on understanding properties related to vulnerability introduction. Currently, we answered our first major research question by discussing the five mostly introduced CVE within Big-Vul. We verified that four out of these five CVEs occurred in the Chromium project. This finding brought us insights to define three hypotheses concerning the reasons why it happens for the Chromium project.

Besides that, we also identified that experienced developers were responsible for introducing the five most common vulnerabilities, which is counter-intuitive since we thought more obvious that inexperienced developers would be responsible for this problem. Our preliminary finding also shows that small commits in terms of addition and deletions are more prone to introduce high severity vulnerabilities, which is also counter-intuitive. However, we need to do further work to better understand the reasons for this to happen.

As future work, we plan to select other case studies to ratify our findings. Thus, first we plan to investigate the Manually-Curated dataset [19] and the VulinOSS dataset [10]. Furthermore, we intend to assess these datasets plus the Big-Vul to better answer our third minor research questions (*RQ1.3*). Besides that, the answers to our second (**RQ2**) and third (**RQ3**) major research questions are crucial to achieve our goal on helping developers to detect software vulnerabilities. Another path of future work is to either validate or reject the three hypotheses (H1, H2, and H3) we defined while answering **RQ1**. Last but not least, we also plan to define more research question with the same purpose of our current goal.

REFERENCES

[1] J. Allen, S. Barnum, R. Ellison, G. McGraw, and N. Mead. 2008. *Software Security Engineering.* Addison-Wesley Professional.

[2] GH Archive. 2021. *GH Archive.* https://www.gharchive.org/

[3] Amiangshu Bosu, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. 2014. Identifying the Characteristics of Vulnerable Code Changes: An Empirical Study. In *International Symposium on Foundations of Software Engineering*. 257–268.

[4] CVE. 2021. *Common Vulnerabilities and Exposures.* https://cve.mitre.org

[5] CVSS. 2021. *The Common Vulnerability Scoring System.* https://nvd.nist.gov/vuln-metrics/cvss

[6] Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, and Damian D. 2014. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*. 92–101.

[7] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 508–512.

[8] GitHub. 2021. *GitHub Rest API.* https://docs.github.com/pt/rest

[9] GitHub. 2021. *GitHub Search.* https://github.com/search

[10] Antonios Gkortzis, Dimitris Mitropoulos, and Diomidis Spinellis. 2018. VulinOSS: A Dataset of Security Vulnerabilities in Open-source Systems. In *Proceedings of the 15th International Conference on Mining Software Repositories*. 18–21.

[11] Jürgen Graf, Martin Hecker, and Martin Mohr. 2013. Using JOANA for Information Flow Control in Java Programs - A Practical Guide. In *Work. Conf. Program. Languages*. 123–138.

[12] Ivan Victor Krsul. 1998. *Software Vulnerability Analysis.* Ph.D. Dissertation. Purdue University.

[13] Bingchang Liu, Guozhu Meng, Wei Zou, Qi Gong, Feng Li, Min Lin, Dandan Sun, Wei Huo, and Chao Zhang. 2020. A Large-Scale Empirical Study on Vulnerability Distribution within Projects and the Lessons Learned. In *Proceedings of the 42nd International Conference on Software Engineering*. 1547–1559.

[14] G. McGraw. 2006. *Software Security: Building Security In.* Addison-Wesley Professional.

[15] Andrew Meneely, Harshavardhan Srinivasan, Ayemi Musa, Alberto Rodriguez Tejeda, Matthew Mokary, and Brian Spates. 2013. When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*. 65–74.

[16] Andrew Meneely, Alberto C. Rodriguez Tejeda, Brian Spates, Shannon Trudeau, Danielle Neuberger, Katherine Whitlock, Christopher Ketant, and Kayla Davis. 2014. An Empirical Investigation of Socio-technical Code Review Metrics and Security Vulnerabilities. In *Proceedings of the 6th International Workshop on Social Software Engineering*. 37–44.

[17] NVD. 2021. *National Vulnerability Database.* https://nvd.nist.gov/

[18] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. 2015. VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits Henning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 426–437.

[19] Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cedric Dangremont. 2019. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *Proceedings of the 16th International Conference on Mining Software Repositories*. 383–387.

[20] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. 2012. A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles. In *Proceedings of the 34th International Conference on Software Engineering*. 771–781.

[21] GH Torrent. 2021. *GH Torrent.* https://ghtorrent.org/