A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries

Jiahao Fan, Yi Li, Shaohua Wang*
SPACE Lab, Informatics
New Jersey Institute of Technology
{jf449,yl622,davidsw}@njit.edu

Tien N. Nguyen
CS Department
The University of Texas at Dallas
tien.n.nguyen@utdallas.edu

ABSTRACT

We collected a large C/C++ code vulnerability dataset from open-source Github projects, namely **Big-Vul**. We crawled the public Common Vulnerabilities and Exposures (CVE) database and CVE-related source code repositories. Specifically, we collected the descriptive information of the vulnerabilities from the CVE database, e.g., CVE IDs, CVE severity scores, and CVE summaries. With the CVE information and its related published Github code repository links, we downloaded all of the code repositories and extracted vulnerability related code changes. In total, Big-Vul contains 3,754 code vulnerabilities spanning 91 different vulnerability types. All these code vulnerabilities are extracted from 348 Github projects. All information is stored in the CSV format. We linked the code changes with the CVE descriptive information. Thus, our Big-Vul can be used for various research topics, e.g., detecting and fixing vulnerabilities, analyzing the vulnerability related code changes. Big-Vul is publicly available on Github.

CCS CONCEPTS

• **Security and privacy**;

KEYWORDS

Common Vulnerabilities and Exposures, Code Changes, C/C++ Code,

ACM Reference Format:
Jiahao Fan, Yi Li, Shaohua Wang and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *17th International Conference on Mining Software Repositories (MSR '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3379597.3387501

1 INTRODUCTION

Vulnerability detection and fixing has been the core and critical activity in software industry. Undetected vulnerabilities can be exploited by hackers and may cause a great loss to users. For example, a new Windows spoofing vulnerability (CVE-2020-0601[1]) that can affect millions of Windows computers has been discovered [9]. Hackers could exploit the vulnerability to decrypt confidential information when users make connections to the affected software. Recently, the detection of security vulnerabilities has been a crucial topic of interest to the research community [3, 4, 7, 10, 11]. However, most studies have only conducted vulnerability detection for certain types of vulnerabilities mainly due to the lack of readily available datasets.

Inspired by the need, we curated a large C/C++ vulnerability dataset, namely **Big-Vul**, from the Common Vulnerabilities and Exposures (CVE) database [1] and open-source projects. First, we crawled the public CVE database to collect all of the available descriptive information of a CVE, e.g., the CVE ID, the CVE severity score, the CVE summary, and references linking to the affected products. Second, through the CVE references, we dug into the relevant products with git open source repositories. Using CVE IDs, we identified vulnerability-related code commits and extracted relevant code changes. In total, our Big-Vul contains 3,754 code vulnerabilities collected from 348 open source projects spanning 91 different vulnerability types. We linked the code changes with the CVEs (including descriptive information). Our dataset Big-Vul can enable the following key analysis of vulnerabilities, but not limited to, (1) deep analysis on the characteristics of different vulnerabilities and vulnerability code changes; and (2) improving the detection and fixing of code vulnerabilities.

Unlike existing work, **Big-Vul** has the following key features:

(1). Zhou et al. [11] constructed a vulnerability dataset by filtering out commits on GitHub using security-related keywords. Unlike their dataset, **Big-Vul** was constructed by utilizing and linking the CVE database, the project bug reports, and the code commits, which helps to improve the accuracy of identifying the vulnerability-related commits with code changes. In addition, they did not label the vulnerability types and only released part of their dataset.

(2). Ponta et al. [5] manually curated a Java vulnerability dataset containing CVE-IDs and code commit IDs. Our Big-Vul contains more information of CVEs, including 21 features, e.g., code changes, CVE summaries, and security scores. Also there are 3754 vulnerabilities in our dataset whereas only 624 vulnerabilities in theirs.

(3). VulinOSS [2] mainly contains the vulnerability information relevant to the project meta-data, e.g., releases versions and code metrics. Our Big-Vul is more code-centric. We processed the source code of projects and their relevant code commits to extract vulnerable functions and their corresponding fixes.

The contributions of this paper are:

[A. Dataset.] We collected and published a large dataset that contains code changes and CVE summaries derived from the CVE database and open source project repositories.

*Corresponding Author
[1]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-0601

MSR '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7517-7/20/05…$15.00
https://doi.org/10.1145/3379597.3387501

[B. The collection process with scripts.] We published our data collection approach with supporting scripts. Our **Big-Vul** with scripts are available here [6].

2 DATA COLLECTION

We built our Big-Vul dataset in the following five steps:

S1. We crawled all of the vulnerability entries in the CVE database, such as the descriptive information for each vulnerability. Specifically, we created a script using BeautifulSoup[2] to parse the web pages of CVE Details and traverse the pages by years. For each CVE entry, we collected the following information: access complexity, authentication required, availability impact, confidentiality impact, CWE ID, CVE ID, CVE summary, integrity impact, publish date, security score, update date and vulnerability classification.

S2. We automatically selected the CVE entries that have reference links of publicly available Git repositories. We only kept the Git repositories that have a clear and fixed traversal path that can lead to the actual code commits. In our dataset, we focused on the Github repositories and some popular products with their own Git server, such as Google Android. For example, when processing the Google Chrome related entries, the CVE reference links lead us to specific *Stable Channel Update for Desktop* pages, such as the page[3] of March 31, 2020. The page contains the CVEs and their associated bug IDs. We retrieved the bug IDs and their corresponding CVEs. Then, we used the bug IDs to identify relevant code commits from the Chrome's mirror repository on Github. Some big popular products, like Chrome and Android, may have their own release pages with security information, while some others may directly link the code repositories to the CVE database as reference links. We developed distinct crawling strategies according to the different structures of pages eventually leading to code repositories.

S3. Each commit is considered as a mini-version of a project. We used the commit IDs to request commit histories of the projects, and mapped each mini-version to the corresponding CVE entries. For each relevant commit, we extracted the code changes between before and after fixing a vulnerability. Finally, we used the code changes information to recover the vulnerable version of a method. Thus we collected the following information for a project: vulnerable methods with their fixes and non-vulnerable other methods.

All of the aforementioned details, such as the name of each project and the details of CVSs, have been stored in a CSV format with a clear structure, and source code functions are zipped into a package. The dataset together with the scripts used for its construction are made publicly available on GitHub [6].

3 DATA DESCRIPTION

Our Big-Vul dataset contains the details of CVE entries from 2002 to 2019. We collected 21 features for each CVE entry. Table 1 describes each CVE feature and its corresponding column name in our CSV file. Our Big-Vul dataset is released in a comma-separated values(CSV) format. We also provide example codes that shows how to manipulate and analyze our data.

Our Big-Vul dataset covers 348 different projects that are linked to 4,432 unique code commits. The 4,432 code commits contain the code fixes for 3,754 vulnerabilities in 91 CWE types. We use the following Figures and Tables to represent Big-Vul.

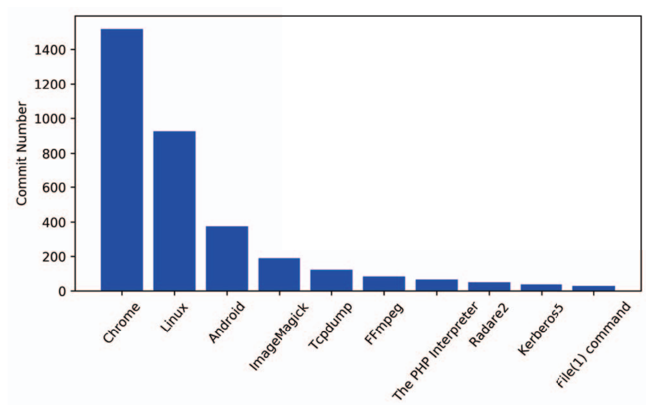

Figure 1: Number of commits for top 10 projects.

Figure 1 shows the number of commits containing fixes for vulnerabilities for 10 different projects with the most C/C++ code commits in Big-Vul. Specifically, the Google Chrome has the most commits, i.e., 1,518, for fixing vulnerabilities. The Linux has the second most commits, i.e., 927 and the Google Android has the third most commits, i.e., 376. The number of the above-mentioned three products, Google Chrome, Linux, and Google Android accounts for 63.65% of the total collected commits from the 348 products. The top 10 projects have a total of 3,399 code commits, which accounts for 76.69% of the total collected commits in Big-Vul.

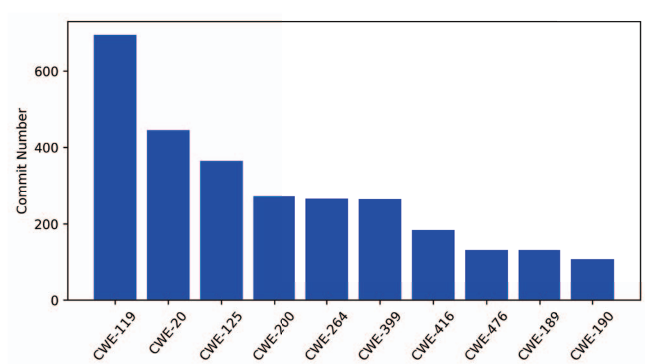

Figure 2: The number of commits for top 10 CWEs.

Figure 2 shows the number of code commits for the 10 top CWE types with the most code commits. The top three CWE types – CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer), CWE-20 (Improper Input Validation) and CWE-125 (Out-of-bounds Read) – are all about data management in C/C++ code and account for 33.94% of the total commits.

[2] https://www.crummy.com/software/BeautifulSoup/

[3] https://chromereleases.googleblog.com/2020/03/stable-channel-update-for-desktop_31.html

Table 1: The Description of CVE Features.

Features	Column Name in the CSV	Description
Access Complexity	access_complexity	Reflects the complexity of the attack required to exploit the software feature misuse vulnerability
Authentication Required	authentication_required	If authentication is required to exploit the vulnerability
Availability Impact	availability_impact	Measures the potential impact to availability of a successfully exploited misuse vulnerability
Commit ID	commit_id	Commit ID in code repository, indicating a mini-version
Commit Message	commit_message	Commit message from developer
Confidentiality Impact	confidentiality_impact	Measures the potential impact on confidentiality of a successfully exploited misuse vulnerability
CWE ID	cwe_id	Common Weakness Enumeration ID
CVE ID	cve_id	Common Vulnerabilities and Exposures ID
CVE Page	cve_page	CVE Details web page link for that CVE
CVE Summary	summary	CVE summary information
CVSS Score	score	The relative severity of software flaw vulnerabilities
Files Changed	files_changed	All the changed files and corresponding patches
Integrity Impact	integrity_impact	Measures the potential impact to integrity of a successfully exploited misuse vulnerability
Mini-version After Fix	version_after_fix	Mini-version ID after the fix
Mini-version Before Fix	version_before_fix	Mini-version ID before the fix
Programming Language	lang	Project programming language
Project	project	Project name
Publish Date	publish_date	Publish date of the CVE
Reference Link	ref_ink	Reference link in the CVE page
Update Date	update_date	Update date of the CVE
Vulnerability Classification	vulnerability_classification	Vulnerability type

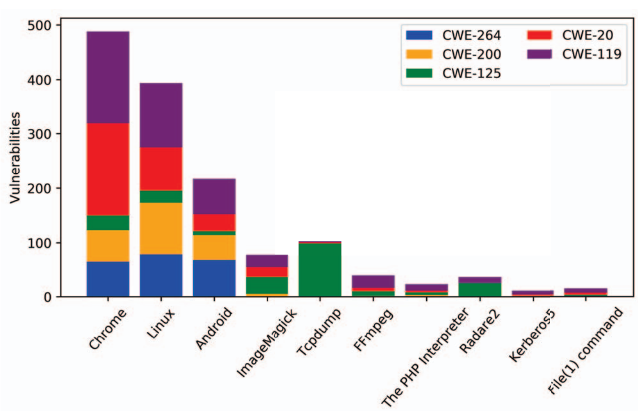

Figure 3: The number of vulnerabilities for top 10 projects in top 5 CWE types.

Figure 3 shows the number of vulnerabilities of the top 10 projects for the top 5 CWE types. Distinct types of CWEs dominate various projects differently. We found that the three CWE types with the most commits, CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer), CWE-20 (Improper Input Validation) and CWE-125 (Out-of-bounds Read), also appear in every top-10 project. For the Google Chrome, the CWE types – CWE-119 and CWE-20 – are the main types, while CWE-125 is the main CWE type for the products Tcpdump and Radare2.

Table 2: Descriptive statistics of Big-Vul.

Measurement	Value
Number of Projects	348
Number of CWE IDs	91
Number of CVE IDs	3754
Number of Commits	4432
Number of Modified Files	8143
Number of Vulnerable Functions	11823
Number of Non-vulnerable Functions	253096

Table 2 shows more statistics of our Big-Vul at code function level. We identified 4,432 code commits relevant to the vulnerabilities spanning 91 CWE types. For a given vulnerable function with the related commits, we kept the vulnerable version of the function and its code changes for fixing the vulnerability In total, we obtained 8,143 modified files, 11,823 vulnerable functions, and 253,096 non-vulnerable functions in our Big-Vul.

Furthermore, we also wanted to study the distribution of vulnerable functions across different projects. Figure 4 shows the number of vulnerable functions for each top top-10 project that has the

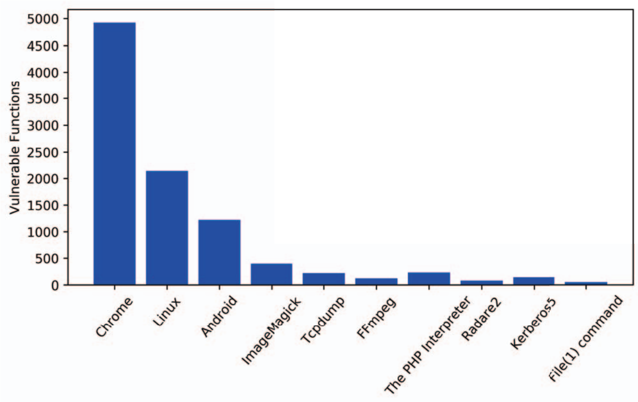

Figure 4: The number of vulnerable functions in top 10 projects.

most commits. Specifically, the Google Chrome, Linux and Google Android are the top three projects that have the most vulnerable functions. They have 4,932, 2,140 and 1,224 vulnerable functions, respectively, which accounts for 70.17% of the total number of vulnerable functions in Big-Vul.

4 DATA APPLICATION

Our Big-Vul dataset can be used for many vulnerability related research areas, e.g., deep understanding CVEs and code changes, code-centric vulnerability detection and the identification of vulnerability fixing patches.

The deep analysis on CVEs and code changes. The collected CVE IDs, CVE summaries and some other detailed CVE information can help conduct a deep analysis on the key features of vulnerabilities using text mining and NLP. Also, our collected code changes can be used to extract code features and give more insights how a vulnerability was fixed. Furthermore, the detailed CVE information is linked with its corresponding code changes, which makes it possible to conduct any analysis with the combination of CVE information and code. The analysis of CVE textual description, related code changes, and their relations can help contribute to the research on explaining a possible code fix for a vulnerability to improve the detection of vulnerabilities.

Code-centric vulnerability detection. In the evolution of software engineering, there is a continuing tension between the need to develop new features and detect vulnerabilities. To relieve developers from the tedious and time-consuming task of manual vulnerability detection, researches on automatic vulnerability detection have been conducted [3, 7, 11]. Our Big-Vul dataset contains code changes for vulnerability fixes so it can be utilized to model code fixes at different levels, such as file, function, and line levels. Using that code-centric information, researchers can abstract the features of code vulnerabilities to define rules, or even train neural network models to learn the code features for detecting vulnerabilities.

Identification of vulnerability fixing patches. For open source projects, due to the publicly availability of their code repositories and commit logs, the vulnerabilities in projects may be exposed to security attacks during the time gap between the fix of a vulnerability and the release of the security version after the fix. In this case, it is important to have a code-changes tracking system geared towards automatic vulnerability fixing patches identification to aid the developer in the management of version release [8, 12]. With the code changes, our Big-Vul can be utilized to do research on vulnerability fixing patches identification.

5 LIMITATION

In our Big-Vul dataset, the CSV file has 306 rows, and the rows related to the Chrome project miss out some descriptive information of some CVEs, e.g., the CVE IDs, CWE IDs, etc. because we used the bug IDs in the official released bug reports by Google as the keywords to retrieve the Chrome mirror repository on GitHub and extract the relevant fix commit information. A tiny number of them were not assigned with CVE IDs. Therefore, we failed to map these entries to CVE database. We will update our dataset with related information once these entries are assigned with CVE IDs, CWE IDs or other important features.

6 RELATED WORK

There are several existing vulnerability datasets created by previous studies [2, 5, 11]. Zhou et al. [11] collected a C vulnerability dataset by filtering out commits on GitHub using security-related keywords. Then they manually checked each commit if it is vulnerable one. Ponta et al. [5] monitored the NVD database and more than 50 different project-specific websites for new vulnerability disclosures by manually checking the available information and extracting the corresponding fix commits. Gkortzis et al. [2] collected a dataset by crawling data from the NVD database and recording the project version information from the database. Then they mapped project versions to the version references (commit tags and branches) found in the corresponding project repositories.

7 CONCLUSION

We present a C/C++ code vulnerability dataset, namely **Big-Vul**, that contains important information such as CVE IDs, CVE severity scores, CVE summaries, mini-versions, code changes, etc. Our Big-Vul dataset were collected from Common Vulnerabilities and Exposures database and the official project bug reports, which means that our dataset is accurate in terms of whether the code changes that mapped to the CVE descriptive information are really vulnerability-related. Containing the mini-versions before and after the vulnerability-related fixes, our dataset can be used to conduct vulnerability-related research by extracting the code changes between the two mini-versions.

In our future work, we will attempt to mine more repositories that use other issue-tracking and source control management systems, such as Mercurial, Subversion, JIRA, and Bugzilla, etc. , instead of only Git, to make our data cover more projects, more vulnerability types, and more programming languages.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) grants CCF-1723215, CCF-1723432, TWC-1723198, CCF-1518897, and CNS-1513263.

REFERENCES

[1] CVE Details. 2020. CVE Details Website. http://https://www.cvedetails.com/.

[2] 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.

[3] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv preprint arXiv:1801.01681* (2018).

[4] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. 2018. Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 449–460.

[5] Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. 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)*. IEEE, 383–387.

[6] This Project. [n.d.]. Our C/C++ dataset. https://github.com/ZeoVan/MSR_20_Code_Vulnerability_CSV_Dataset.

[7] Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. 2018. Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 757–762.

[8] Antonino Sabetta and Michele Bezzi. 2018. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 579–582.

[9] Zack Whittaker. 2020. Microsoft and NSA say a security bug affects millions of Windows 10 computers. https://techcrunch.com/2020/01/14/microsoft-critical-certificates-bug/.

[10] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. 2014. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 590–604.

[11] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems*. 10197–10207.

[12] Yaqin Zhou and Asankhaya Sharma. 2017. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 914–919.