CrossVul: A Cross-Language Vulnerability Dataset with Commit Data

Georgios Nikitopoulos
University of Thessaly
gnikitopoulos@inf.uth.gr

Konstantina Dritsa
Athens University of Economics and Business
dritsakon@aueb.gr

Panos Louridas
Athens University of Economics and Business
louridas@aueb.gr

Dimitris Mitropoulos
University of Athens
dimitro@ba.uoa.gr

ABSTRACT

Examining the characteristics of software vulnerabilities and the code that contains them can lead to the development of more secure software. We present a dataset (~1.4 GB) containing vulnerable source code files together with the corresponding, patched versions. Contrary to other existing vulnerability datasets, ours includes vulnerable files written in more than 40 programming languages. Each file is associated to (1) a Common Vulnerability Exposures identifier (CVE ID) and (2) the repository it came from. Further, our dataset can be the basis for machine learning applications that identify defects, as we show in specific examples. We also present a supporting dataset that contains commit messages derived from Git commits that serve as security patches. This dataset can be used to train ML models that in turn, can be used to detect security patch commits as we highlight in a specific use case.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**; **Software post-development issues**.

KEYWORDS

Dataset, vulnerabilities, security patches, commit messages

ACM Reference Format:
Georgios Nikitopoulos, Konstantina Dritsa, Panos Louridas, and Dimitris Mitropoulos. 2018. CrossVul: A Cross-Language Vulnerability Dataset with Commit Data. In *Proceedings of The 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

At the constantly evolving forefront of security, vulnerability identification remains a critical and challenging topic. Vulnerability-exploitation can lead to unauthorized breaches and cause substantial financial or social loss. There are numerous approaches to detect vulnerabilities spanning from static analysis [12, 13] and dynamic analysis [8] to recent advances in Machine Learning (ML) approaches [7, 9].

At the same time, ML applications are data-hungry and the collection of high-quality vulnerable training samples of source code remains a challenge. On this data curation topic, many attempts have employed synthetic source code data in order to train models, ensuring the quantity of training samples but losing in quality, as artificial code examples lack the naturalness and complexity of real-world scenarios. Other attempts leverage data derived from open-source repositories by labeling vulnerable source code using program analysis tools. These approaches may produce datasets with several false positives.

Motivated by the latest developments, we have curated a dataset of vulnerable source code files and their secure counterparts. To do so, we have examined 5877 GitHub commits referenced by NVD (National Vulnerability Database) [3] and CVE (Common Vulnerability and Exposures) [1] entries. Each of these entries contains information about a certain security vulnerability or exposure. Further, it includes data about the issue such as a text description, the vulnerability type as defined by the Common Weakness Enumeration identifier (CWE ID) [1], the severity of the issue by means of CVSS (Common Vulnerability Scoring System) [2] Score, and other data such as references.

For each commit we locate (1) the files that are related to the security patch and (2) the vulnerable ones, and label them accordingly. The information that we store for each file includes its original repository, commit identifier, filename, its associated vulnerability type and CVE identifier and finally its associated commit message. Commit messages are kept in a separate, supporting dataset that can be used to train Natural Language Processing text classification models to detect commits that contain security patches.

Unlike existing work, our dataset spans across multiple programming languages, with files coming from a large collection of repositories. Vulnerable and non-vulnerable code is paired in code changes and is accompanied by the respective commit message, making it useful for ML applications.

The contributions of our work include: a) the construction process of a labeled dataset that contains 27476 files (1.4 GB in total), b) the dataset, and c) how it can be used to produce research results. Our dataset[1] and source code[2] are available on Zenodo. A demonstration of CrossVul and how to use it can be found on YouTube.[3]

2 DATASET CONSTRUCTION PROCESS

Figure 1, illustrates the steps of the dataset construction process. First, we collected all NVD entries that provided references to corresponding GitHub projects. We kept the references to GitHub

[1]https://doi.org/10.5281/zenodo.4734050
[2]https://doi.org/10.5281/zenodo.4741963
[3]https://www.youtube.com/watch?v=sz9z2Zul2aw

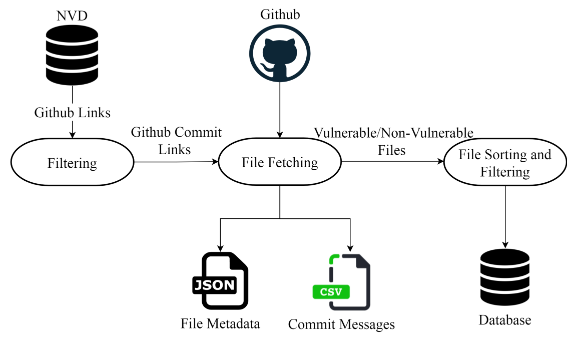

Figure 1: The dataset construction process.

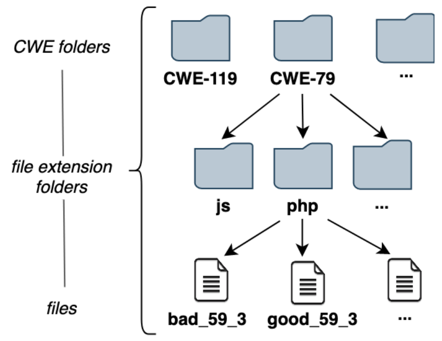

Figure 2: File hierarchy system.

commits as is, and processed references that pointed to pull requests to retrieve all relevant commits. Invalid links such as broken ones or links that pointed to branches and not specific commits or pull requests were filtered. Then, we identified the individual files that were modified in each commit using the `git-diff` command. Furthermore, we retrieved the file containing the security patch and the file without the patch (related to the previous commit). We labeled the files as secure and vulnerable respectively. Files that contained test cases were also filtered (i.e. we dropped the files with the word "test" in their filename). Finally, we sorted the collected files in a specific file hierarchy based on the CWE ID and the language.

As we discussed earlier, we have also generated a commit message dataset. This dataset was populated during the second step of the aforementioned process. We use the security patch commit list to collect and label corresponding commits. An equal amount of generic commit messages that are not related to security patches were also collected. Such messages can be utilized as counterexamples in a Natural Language Processing text classification setting.

Specifically, for each security patch, we obtain a random commit sourced from its repository. This significantly avoids the chance of accidentally selecting security patch commits since they average only the 0.8% of all commits in a repository according to our dataset.

Table 1: Descriptive statistics measurements for our datasets

Measurement		Value		
Main dataset				
GitHub projects		1675		
Commits		5877		
Unique CWEs		168		
Unique CVEs		5131		
All files		27476		
Vulnerable files		13738		
Non-vulnerable files		13738		
Unique file extensions		48		
Supporting dataset				
		Min	*Max*	*Avg*
Security-patch commits	Word count	0	2169	53.75
	Character count	2	12201	327.56
Random Generic commits	Word count	0	6113	29.25
	Character count	2	46501	179.18

3 DATASET DESCRIPTION

Our dataset contains: a) the directory with all source code files, b) a JSON file containing commit and file information associated with CVE IDs and CWE IDs, c) the commit message dataset which consists of three separate CSV files. Figure 3 illustrates the structure of the JSON file.

Table 1 provides descriptive statistics regarding our dataset. Specifically, the dataset contains 1.4 GB of source code including 27476 files collected from 1675 GitHub repositories. Half of these files contain vulnerabilities and the other half are the respective patched versions of the vulnerable source code. As described in the previous section, our dataset follows a specific structure: we group all files per vulnerability and in the corresponding folder we group them per language. In both cases grouping is easy to perform using the defect ID and the file extension respectively. This allows researchers to quickly retrieve files that are written in a specific language and are related to a particular class of vulnerabilities. Overall, the included vulnerabilities span over 168 unique CWE categories and 5131 unique CVEs. Table 2 shows the top five vulnerabilities in terms of appearance in the dataset.

Figure 2 illustrates the hierarchy of our dataset. Root directories are named after the CWE ID (<CWE_ID>). Inside each directory we include directories with files written in the same programming language. The names of these directories come from the extensions of the files that contain (<filename_extension>); we use "None" when the language cannot be determined. In particular cases multiple

Table 2: Five most represented vulnerability types in dataset based in terms of appearance.

CWE ID	Vulnerability Description	Files	Commits
CWE-79	Cross-site Scripting	4094	813
CWE-20	Improper Input Validation	2028	462
CWE-125	Out-of-bounds Read	1958	398
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	1724	468
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	1334	332

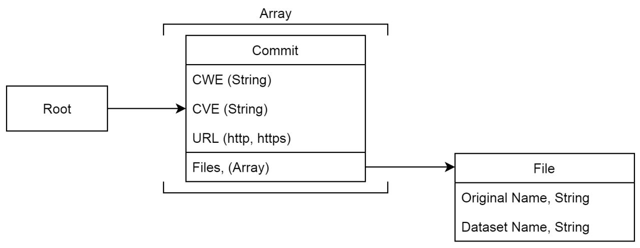

Figure 3: The dataset metadata JSON file structure.

folders may contain files written in the same language (e.g., `.h` and `.c` folders). Each source code file name follows the format: "*good/bad_commitID_fileID*". We explain each part in the following:

- The literal string "*bad*" or "*good*" denotes if the source code file is vulnerable or secure respectively.
- The commit ID is a numerical identifier for the commit entry associated with the source code file. Each commit can be traced through its ID in the `file_info.json` file. Figure 3 illustrates the structure of this file. Note that all the files that where modified in the same commit have the same commit ID in their filename.
- The file ID is a numerical identifier used to distinguish between different files of the same commit. Furthermore, the vulnerable and secure versions of the same file have the same file ID, e.g., "*bad_59_3*" and "*good_59_3*".

Our method ensures that no filename collisions will occur when merging different directories that contain files associated with different vulnerability types. This allows researchers to group many different vulnerability categories together into fewer, more general categories. For instance, one could group the CWE-121 and CWE-122 categories together to make a category for both stack and heap-based overflows. Additionally, one can quickly examine the most popular defects in a language. Figure 4 depicts the distribution of files across the five most frequent vulnerability types and popular programming languages (i.e. with the most files containing code written in the corresponding language). Observe that "*Improper Restriction of Operations within the Bounds of a Memory Buffer*" (CWE-119) is the most popular defect among C files.

Furthermore, the supporting dataset includes 11744 commit messages that correspond to 5872 commits of the main dataset plus an equal amount of generic random commit messages sourced from the same repositories. The supporting dataset consists of two CSV files (in `QUOTE_ALL` mode). The first is related to the commit messages coming from security patch commits and contains four columns, namely: vulnerability type, CVE Identification, commit URL and the commit message itself. The second file contains generic commit messages that are not related to security patches. Table 1 shows that the average length of the security-patch commits is larger than the length of generic commits. This can be explained by the fact that security patches can have more information concerning the vulnerability and the repairing process.

Figure 5 illustrates the distribution of the ten most frequent vulnerabilities in the ten most vulnerable repositories (i.e., the repositories containing the most defects). The "*Unknown*" CWE category includes CVEs that are not assigned in any of the existing CWE

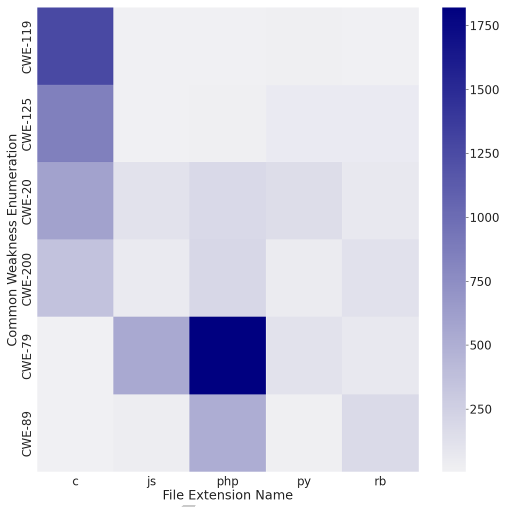

Figure 4: Distribution of the five most frequent CWEs and popular programming languages.

categories. Such cases can be found in the "*None*" folder at the CWE level of the dataset. An interesting observation is that Linux has by far the most commits concerning popular CWEs such as CWE-119 and CWE-200. In the majority of the remaining projects, CWE-79 holds the largest percentage. Notably, in the case of the *tcpdump* project the CWE-125 ("*Out-of-bounds Read*") is the most popular.

4 APPLICATIONS

Our dataset can be used for a variety of tasks in ML applications.

Vulnerability detection and code repair. Recently there have been several advances in the field of deep learning [13, 15] for vulnerability detection. Our dataset offers a wide variety of files, across multiple programming languages, collected from real-world scenarios and from a wide range of repositories. Furthermore, it contains vulnerable files paired with secure counterparts offering a balanced dataset for training deep learning models. Such features make it useful not only for the tasks of vulnerability detection and classification but also for training deep learning models on the task of automatic source code correction.

Code change embeddings. Another utilization may involve learning vector representations of the actual code changes, in a way that the vectors of similar vulnerability patches would be close in the vector space [6]. These representations can then be used for a variety of tasks such as the detection of vulnerability patch commits or their classification to vulnerability categories.

Applications on commit messages. The commit messages that accompany the code changes of our dataset can be utilized in a variety of tasks. First, they can be used to fine-tune the aforementioned code change embeddings, thus adding the aspect of semantic intent of the vulnerability patch, resulting in higher-quality representations [6]. Furthermore, the code change representations paired with their respective commit messages can be used to train a deep learning model to automatically generate commit messages. Finally, they can be used for the detection and classification of vulnerability commit messages and facilitate the creation of new source code vulnerability datasets.

As an example, consider Table 3. The table, demonstrates the performance metrics of three classification ML models against the commit message dataset, on the task of commit message classification to either "security-patch" or "non security-patch" commit messages. The Naive Bayes and Random Forest models use a *tf-idf* vectorizer while the Convolutional Neural Network model uses one-hot encoding. We pre-processed the messages by stemming words using the Lancaster Stemmer algorithm [10] and by removing stop-words for the cases of the Naive Bayes and Random Forest models. Our initial findings indicate that Random Forest models perform slightly better than the rest of the models in identifying security patch commit messages.

5 LIMITATIONS

One limitation of our dataset is that it is split and labeled in files, not functions. This reduces the number of data samples and subsequently the dataset's granularity. A potential solution would be to use a language parser for each programming language appearing in the dataset to split the files into functions and then label them using the `git-diff` of the commits.

An additional limitation involves the utilization of the commit message dataset as training data for ML models aimed to fully automate vulnerable source code mining. One has to ensure the quality of the resulting dataset to manually inspect the commits classified as security patches. The random sampling that we used for labeling negatives do not correspond to the gold truth. However, the 0.8% of security patches is very low, it is unlikely that we pick false positives in sampling. Moreover, the efficacy of the model in a mining scenario can be further improved by decreasing false positives at the expense of more false negatives by changing the classification decision threshold of models from the default value of 0.5.

6 RELATED WORK

There have been many attempts to create source code vulnerability datasets, especially for utilization in the context of ML applications. The main approaches of data collection in this field are:

Synthetic code creation. In this context, the vulnerability datasets are composed of artificial code examples, generated for some specific purpose, e.g., to test security scanners [14]. Such datasets ensure quantity for deep learning applications [7, 9, 13]. However, on the quality front, the ability of models to generalize over real-world scenarios is limited by the simplicity of the generated code.

Implicitly collected vulnerabilities. In this category, vulnerabilities are identified using implicit methods, such as static analysis tools [12, 13] or by filtering commits using security-related keywords [15, 16]. In this case, the accuracy, variety and overall quality of the data inherit the limits of the selected methodology.

Explicitly collected vulnerabilities. Here, vulnerable source code is collected based on publicly disclosed vulnerability incidents. Gkortzis et al. [5] present a dataset containing the repositories of vulnerable open-source systems along with vulnerability metadata. However, the label granularity is on project-level (in our case is on file-level). Jiahao et al. [4] created a collection of vulnerable C code by crawling the CVE database and extracted vulnerability-related code changes of methods, complemented by the respective CVE descriptions. Ponta et al. [11] manually curated a Java vulnerability dataset ensuring high accuracy but resulting in a comparatively limited collection of 1282 commits mapped to 624 vulnerabilities. However, this dataset is limited to security patch commit links and does not contain any actual source code.

Our work falls in the latter category, ensuring high-quality data with accurate labels and code naturalness. We collected a balanced dataset of vulnerable source code and its patched counterparts in file-level granularity, by crawling the NVD. We also complement our dataset with a supporting dataset of all the commit messages that accompany the collected file changes. Unlike existing work, our dataset spans across multiple programming languages, from a large collection of repositories. Vulnerable and non-vulnerable code is paired in code changes. These features make it useful for a wide variety of tasks in ML applications.

7 CONCLUSIONS

We have presented a dataset of vulnerable source code and its respective patched/secure counterparts retrieved from open source software repository Git commits. We have also obtained the commit messages involved in those commits and trained models to distinguish vulnerability patch related commit messages. This allows an expansible vulnerable source code dataset that can be used for data-driven automated vulnerability detection techniques and other research purposes.

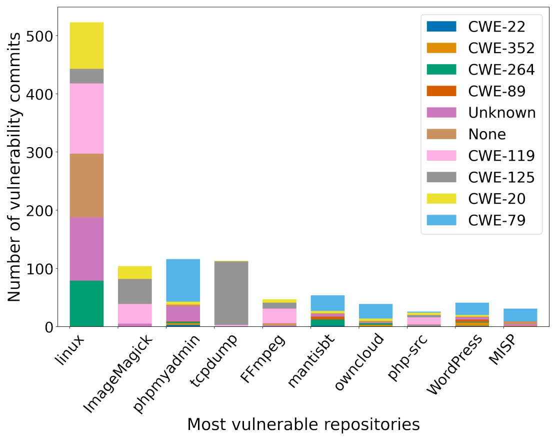

Figure 5: Distribution of ten most frequent CWEs in the ten most vulnerable repositories.

Table 3: Commit message classification performance metrics for different ML models.

ML Model	Accuracy	F1-Score	Recall	Precision
CNN	83%	83%	83%	83%
Naive Bayes	82%	82%	82%	82%
Random Forest	84%	84%	84%	84%

REFERENCES

[1] [n.d.]. CVE - Common Vulnerability and Exposures. https://cve.mitre.org/. [Online; accessed 02-May-2021].

[2] [n.d.]. CVSS - Common Vulnerability Scoring System SIG. https://www.first.org/cvss/. [Online; accessed 02-May-2021].

[3] [n.d.]. NVD - National Vulnerability Database. https://nvd.nist.gov/. [Online; accessed 02-May-2021].

[4] 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* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 508–512. https://doi.org/10.1145/3379597.3387501

[5] 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* (Gothenburg, Sweden) *(MSR '18)*. Association for Computing Machinery, New York, NY, USA, 18–21. https://doi.org/10.1145/3196398.3196454

[6] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. 2020. CC2Vec: Distributed Representations of Code Changes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 518–529. https://doi.org/10.1145/3377811.3380361

[7] Xin Li, Lu Wang, Yang Xin, Yixian Yang, and Yuling Chen. 2020. Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning. *Applied Sciences* 10 (03 2020), 1692. https://doi.org/10.3390/app10051692

[8] Yuekang Li, Yinxing Xue, Hongxu Chen, Xiuheng Wu, Cen Zhang, Xiaofei Xie, Haijun Wang, and Yang Liu. 2019. Cerebro: Context-Aware Adaptive Fuzzing for Effective Vulnerability Detection. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 533–544. https://doi.org/10.1145/3338906.3338975

[9] 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 e-prints*, Article arXiv:1801.01681 (Jan. 2018), arXiv:1801.01681 pages. arXiv:1801.01681 [cs.CR]

[10] Chris D. Paice. 1990. Another Stemmer. *SIGIR Forum* 24, 3 (Nov. 1990), 56–61. https://doi.org/10.1145/101306.101310

[11] Serena E. Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric 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* (Montreal, Quebec, Canada) *(MSR '19)*. IEEE Press, 383–387. https://doi.org/10.1109/MSR.2019.00064

[12] Razvan Raducu, Gonzalo Esteban, Francisco Lera, and Camino Fernández. 2020. Collecting Vulnerable Source Code from Open-Source Repositories for Dataset Generation. *Applied Sciences* 10 (02 2020), 1270. https://doi.org/10.3390/app10041270

[13] Rebecca L. Russell, Louis Kim, Lei H. Hamilton, Tomo Lazovich, Jacob A. Harer, Onur Ozdemir, Paul M. Ellingwood, and Marc W. McConley. 2018. Automated Vulnerability Detection in Source Code Using Deep Representation Learning. *arXiv e-prints*, Article arXiv:1807.04320 (July 2018), arXiv:1807.04320 pages. arXiv:1807.04320 [cs.LG]

[14] Paul E. Black Vadim Okun, Aurelien Delaitre. [n.d.]. Report on the Static Analysis Tool Exposition (SATE) IV. http://dx.doi.org/10.6028/NIST.SP.500-297. Accessed: 2020-8-13.

[15] 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*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., 10197–10207.

[16] 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* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 914–919. https://doi.org/10.1145/3106237.3117771