MoreFixes: a large-scale dataset of CVE fix commits mined through enhanced repository discovery

Akhoundali, J.; Nouri, S.R.; Rietveld, K.F.D.; Gadyatskaya, O.

Citation
Akhoundali, J., Nouri, S. R., Rietveld, K. F. D., & Gadyatskaya, O. (2024). MoreFixes: a large-scale dataset of CVE fix commits mined through enhanced repository discovery. *Promise 2024*, 42-51. doi:10.1145/3663533.3664036

Version:	Publisher's Version
License:	Creative Commons CC BY 4.0 license
Downloaded from:	https://hdl.handle.net/1887/4213015

Note: To cite this publication please use the final published version (if applicable).

MoreFixes: A Large-Scale Dataset of CVE Fix Commits Mined through Enhanced Repository Discovery

Jafar Akhoundali
LIACS, Leiden University
Leiden, The Netherlands
j.akhoundali@liacs.leidenuniv.nl

Sajad Rahim Nouri
Islamic Azad University
Ramsar, Iran
s.rahimnouri@iau.ir

Kristian Rietveld
LIACS, Leiden University
Leiden, The Netherlands
k.f.d.rietveld@liacs.leidenuniv.nl

Olga Gadyatskaya
LIACS, Leiden University
Leiden, The Netherlands
o.gadyatskaya@liacs.leidenuniv.nl

ABSTRACT

Vulnerability datasets have become an important instrument in software security research, being used to develop automated, machine learning-based vulnerability detection and patching approaches. Yet, any limitations of these datasets may translate into inadequate performance of the developed solutions. For example, the limited size of a vulnerability dataset may restrict the applicability of deep learning techniques.

In our work, we have designed and implemented a novel workflow with several heuristic methods to combine state-of-the-art methods related to CVE fix commits gathering. As a consequence of our improvements, we have been able to gather the largest programming language-independent real-world dataset of CVE vulnerabilities with the associated fix commits. Our dataset containing 26,617 unique CVEs coming from 6,945 unique GitHub projects is, to the best of our knowledge, by far the biggest CVE vulnerability dataset with fix commits available today. These CVEs are associated with 31,883 unique commits that fixed those vulnerabilities. Compared to prior work, our dataset brings about a 397% increase in CVEs, a 295% increase in covered open-source projects, and a 480% increase in commit fixes. Our larger dataset thus substantially improves over the current real-world vulnerability datasets and enables further progress in research on vulnerability detection and software security.

We release to the community a 14 GB PostgreSQL database that contains information on CVEs up to January 24, 2024, CWEs of each CVE, files and methods changed by each commit, and repository metadata. Additionally, patch files related to the fix commits are available as a separate package. Furthermore, we make our dataset collection tool also available to the community.

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

This work is licensed under a Creative Commons Attribution 4.0 International License.

PROMISE '24, July 16, 2024, Porto de Galinhas, Brazil
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0675-2/24/07
https://doi.org/10.1145/3663533.3664036

KEYWORDS

Vulnerability dataset, software repository mining, open-source, real-world vulnerability dataset, dataset, CVE

ACM Reference Format:
Jafar Akhoundali, Sajad Rahim Nouri, Kristian Rietveld, and Olga Gadyatskaya. 2024. MoreFixes: A Large-Scale Dataset of CVE Fix Commits Mined through Enhanced Repository Discovery. In *Proceedings of the 20th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '24), July 16, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3663533.3664036

1 INTRODUCTION

Vulnerabilities in Open-Source Software (OSS) can cause massive damage to systems [25]. When software vulnerabilities are disclosed they are assigned a Common Vulnerabilities and Exposures (CVE) identifier and recorded in the National Vulnerability Database (NVD) [26]. CVE records are typically brief and contain information such as a unique identifier, description, severity estimation, and references [6]. CVE entries with links to vulnerability fix commits in the corresponding software project repositories can be used to extract vulnerable and non-vulnerable versions of the relevant code snippets, resulting in the creation of a vulnerability dataset associated with patches.

Such datasets have proven effective for studying vulnerability characteristics [24], identification of security weaknesses [22, 36], and development of vulnerability detection techniques [1, 7, 12], which are a crucial component of modern secure software development processes [23]. Still, the size of currently available high-quality vulnerability datasets is an ongoing challenge, in particular for deep learning models that rely heavily on the training dataset size [17]. For instance, a recent study by Kluban et al. [22] mentioned the lack of good vulnerability datasets and consequently saw the need to create their dataset by combining data from Snyk[1] and VulnCodeDB[2]. Unfortunately, both data sources have been deprecated for approximately two years now and are not being updated.

Several other approaches have been proposed to increase the number of samples in vulnerability datasets. Vulnerable code generation can produce a large number of samples [5, 37, 39]. However, these samples are not similar to real-world data, which puts their usage in real-world vulnerability analysis under question. Another

[1] https://github.com/snyk/vulnerabilitydb
[2] https://www.vulncode-db.com/

approach is to insecurely refactor real-world software to make the vulnerabilities unique and realistic [35]. Although in such cases the control flow of the software is real-world, the artificially introduced vulnerabilities can be different from real-world vulnerabilities.

Thus, mining of fix commits from CVEs in open-source software (OSS) projects is the preeminent way to collect real-world vulnerability datasets. The majority of prior work on such mining is either done manually [30], what certainly limits the size of such datasets, or it is based on detecting Git-commit links to GitHub projects (or any other version control system) [4, 7, 12, 32]. **Our key observations** are that *CVE descriptions in the NVD usually do not have uniform description data and reference types*, and *the fix commits are often not mentioned*. This causes existing mining methodologies to frequently fail to relate a CVE to its fix commits. In this work, we describe how we have enhanced existing state-of-the-art methodologies using a novel workflow to match a CVE to its fix commit in a repository. Our approach has resulted in a significantly larger-scale real-world CVE fix commit dataset. This is achieved by applying several heuristic methods combined with the Prospector tool [34], which can extract relevant candidate fix commits given a CVE and OSS repository.

Our contributions are as follows:

(1) We propose several heuristic methods to detect CVEs related to OSS while removing irrelevant projects.
(2) We design a novel automatic workflow that gathers CVE fix commits, which results in a significantly larger real-world vulnerability dataset compared to prior work. Moreover, our tool implementing the proposed workflow allows importing the dump and updating the data sources to add new vulnerabilities efficiently. We share the tool with the community.
(3) Our dataset and the tool are shared with the community as a PostgreSQL database dump, including patch files[3]. This allows the full reconstruction of vulnerable projects across several programming languages.

2 OUR WORKFLOW

Our goal is to collect CVE fix commits in OSS projects that were previously missed by other approaches. Fig. 1 presents our workflow. As can be seen in the figure, we augment the state-of-the-art CVEFixes workflow [4] with several heuristic methods to detect the relation between a CVE and its corresponding OSS projects. These help identify the corresponding fix commits in case the CVE description does not contain a direct fix commit link. In the remainder of this section, we discuss the different steps from Fig. 1.

Data Sources. To determine sources of CVE information that contain the most suitable references, we manually analyzed several datasets related to CVEs and open-source software such as Snyk[4], OSV[5]. Based on our analysis, the NVD[6] (also used by CVEFixes) and the GitHub Security Advisory (GHSA)[7], which both provide CVE descriptions and references, have been chosen as the main sources of data. In the GHSA, some vulnerability entries are linked

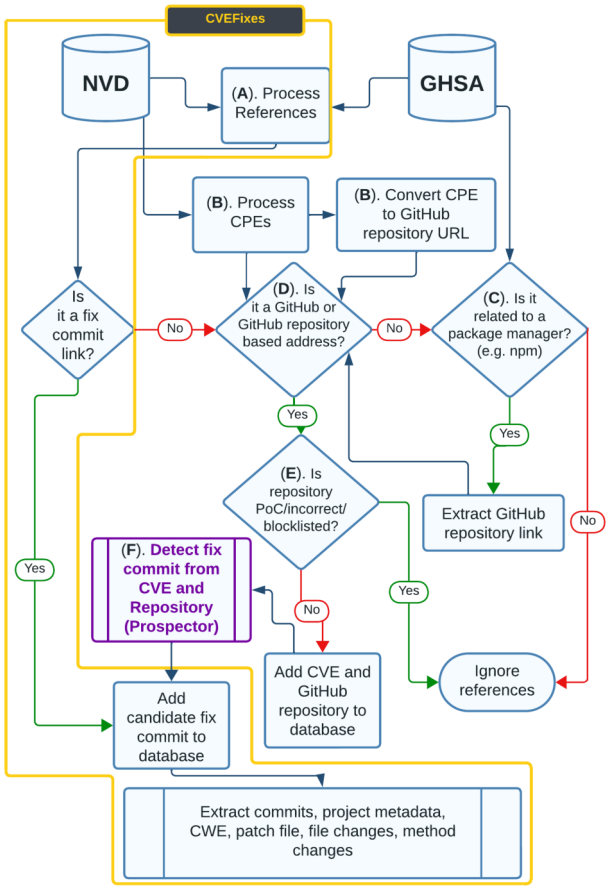

Figure 1: The flowchart diagram of our methodology, compared to the state-of-the-art cross-programming language vulnerability dataset collection method CVEFixes [4] (shown within the yellow contour).

to a relevant repository or a software registry package, as these were reviewed by GitHub staff. This information is also collected.

To improve the detection of CVEs in OSS, we also use the Common Platform Enumeration (CPE)[8] dictionary provided by the NVD. CPE is a structured way to identify different versions of software, which is used in the NVD to list affected products. In addition, similarly to CVEFixes [4], we also collect and store the assigned Common Weakness Enumeration types (CWEs[9]) of each CVE.

2.1 References Processing (A)

A reference in a CVE description is simply a URL. Similar to prior works [4, 7, 12, 32, 42], we consider a direct link to a GitHub commit as the fix commit. Whereas none of the mentioned works does further process a link if it is not a direct commit link, our workflow employs several methods to first find the GitHub repository corresponding to a CVE (discussed in Sec. 2.2 - 2.5), which subsequently is used to extract the vulnerability fix commits (see Sec. 2.6). As a

[3] https://github.com/JafarAkhondali/Morefixes
[4] https://github.com/snyk/vulnerabilitydb
[5] https://github.com/google/osv.dev
[6] https://nvd.nist.gov/
[7] https://github.com/github/advisory-database
[8] https://nvd.nist.gov/products/cpe
[9] https://cwe.mitre.org/

result, we significantly increase the number of collected CVEs that are related to GitHub projects.

2.2 CPE Processing (B)

In NVD, each CVE can be linked to multiple CPEs as the directly affected software. A CPE name comprises a sequence of fields. When a CPE refers to a software package, in many cases the third and fourth components of the CPE name refer to the organization and software name. When available, we try to find the link to the corresponding GitHub repository using the following steps:

- We collect the CPEs referenced by the CVE, and from these CPEs we obtain the organization and software name. We then try to match these with data in the NIST CPE Dictionary to find links.
- If the first step fails, we convert the CPE components "organization_name:software_name" to "https://github.com/organization_name:software_name" and check whether this repository exists.

After the translation of a CPE to a link by either of the mentioned methods, the link will be treated exactly like a link from a reference that was not a direct commit, and it will be passed to step (D), discussed in Sec. 2.4.

2.3 Registry Package Manager (C)

The input to this step is either a URL from step (D) or a package name with its ecosystem from GHSA. In the latter case, depending on the software registry platform, we convert the package name to the URL related to it. For example, for the `express` package from the `npm` ecosystem, this would yield the URL "https://www.npmjs.com/package/express".

Subsequently, both cases can be treated as a URL, and the URL is checked if it belongs to a package registry address and has a corresponding GitHub repository. If it is not possible to use the official software registry platform to extract GitHub repository links with web scraping, we use Open Source Insights[10] to extract the GitHub project link. This mechanism is implemented for several programming languages (JavaScript, Golang, Ruby, PHP, Python, Java, Rust, and C#). If a GitHub link is not found at this stage, the reference is ignored.

2.4 GitHub Repository Address (D)

After processing the links (CVE references, and CPE-based link extractions) in the previous steps, a regular expression (regex) is used to check if the link is related to a GitHub project (e.g. exact project link, link to issue, pull request, tags, etc). If so, the base GitHub project link will be passed to the next step.

2.5 Blocklisted Repositories and Patterns (E)

When a link is identified as a GitHub repository or a GitHub resource, it will first be validated using a specially designed blocklist to make sure the GitHub repository corresponds to a software project. For example, sometimes references in a CVE are links to a proof-of-concept exploit [11] or a repository related to security research. We created the blocklist by manually selecting some keywords (such as "exploit", "0day", "bug bounty", "PoC", "vulnerability", "security", etc.) and some full repository names that are not relevant to the original project. For example, repositories named "Disclosure" or "RVD" were mentioned in the references several times, but if we filter each word as a regex, other valid repositories would be filtered.

To tune the blocklist, we ran the pipeline 5 times and each time we sorted the top repeated GitHub project URLs linked from different CVEs and manually checked repository names and count of the CVEs. After each round of running the pipeline, we fine-tuned the blocklist to remove specific keywords and repositories from the list. In total, we started with 114,432 different links to GitHub projects, and we blocked 36,835 links (32%) using the final blocklist version. Among those, we blocked 18,446 links in the NVD dataset, 10,515 links from GHSA references, and 7,874 links extracted from the CPE dictionary provided by the NVD dataset.

Note that there might still be some irrelevant repositories or keywords that we did not block. This irrelevant data will then be analyzed by the remainder of the pipeline that incorporates further quality checks as we discuss next.

Table 1: Examples of Prospector commit rules and their respective scores [34].

Score	Description
64	Commit message mentions the vulnerability identifier
32	Commit message mentions a GitHub issue or bug-tracking ticket containing the vulnerability ID
8	Commit modifies files mentioned in the advisory
2	Commit message mentions a bug-tracking ticket

2.6 Detecting Candidate Fix Commits (F)

To detect candidate fix commits, we use the state-of-the-art Prospector tool [34] that can detect and score candidate vulnerability fix commits given the repository URL and CVE identifier [34]. Prospector analyzes a given CVE description and Git project using a set of handcrafted rules, such as a mention of CVE-ID in the commit message or a commit touching files mentioned in the advisory, etc. Each rule is assigned a score reflecting how likely the matching commit is to be a fix commit for the given CVE. For illustrative purposes, we provide 4 representative examples of Prospector rules with their scores in Table 1; the full list of rules and scores is available in [34]. Commits that are near the time frame of the CVE publication date are collected and analyzed using these rules, and each candidate commit receives a total score from Prospector corresponding to the sum of scores for each matched rule. Subsequently, Prospector reports and ranks possible candidate commits for each CVE with their matched rules and scores.

Despite the good results from Prospector, its performance (in terms of execution time) does not allow all CVEs associated with GitHub projects to be processed in a reasonable time. Additionally, there appears to be a limit for the maximum number of commits to scan with the original Prospector implementation, and when this limit is reached, the tool crashes. For example, scanning for CVE-2021-28952 in the Linux kernel with the default configuration of 2,000 commits to scan will fail, as the total count of commits to scan is around 23,000. When we modified Prospector to scan only 2,000

[10] https://docs.deps.dev/

commits, this CVE-ID took 4 hours and 32 minutes to complete, with an average scanning speed of 7 commits per minute. This and other reported experiments were done on a server machine with AMD EPYC 7282 CPU @2.8 GHz with 64 GB of RAM running Ubuntu 22.04 as the operating system

Due to these practical limitations, we updated Prospector and improved its scalability as follows:

- **Parallel processing:** Combinations of a CVE and a GitHub repository URL were fed to Prospector in parallel while preventing race conditions. As Prospector uses the Git command, we allowed only one process to work on each CVE for each repository simultaneously.
- **Caching:** Cloning projects (especially large projects like the Linux kernel) was time-consuming and repeating, so we cached repositories to increase performance. When the machine is running out of storage, the cached repositories are removed automatically.
- **Tag detection:** Prospector's tag detection was found to be very time-consuming. Removing this part increased the performance significantly, and we did not observe any change in candidate output ranking.
- **Chosen candidates:** We found that the root cause behind the crashing when the initial count of candidate commits to scan is higher than a certain threshold was that Prospector decides to stop, rather than continuing with a limited set of candidate commits. In such cases, we modified Prospector to only process the same amount of commits as the threshold, with priority given to commits that are near the CVE publication date and to commits that were mentioned in the CVE references. In our work, we used 2,000 as the threshold (candidate commits limit) for running the Prospector.

With our optimizations for commit processing, the aforementioned CVE-2021-28952 from the Linux kernel took only 8 minutes to complete scanning, resulting in a 97% decrease in runtime with an average speed of 250 commits per second scan.

In total, 77,597 CVEs were mapped to GitHub projects. The process to execute Prospector on all these CVEs mapped to GitHub projects took ≈9 days to complete. For each CVE repository link processed by Prospector, we store the top 10 commits with the highest scores in the database. However, not all these candidate commits might be related to the CVE, and we further discuss our approach to address this.

Choosing the Score Threshold. To be able to automatically choose the most relevant fix commits only, a minimum threshold Prospector score needs to be established. To this end, we chose to manually review a set of samples by labeling each CVE and commit pair as *related* or *not related*. A single CVE may have multiple fix commits, and also a single fix commit may fix multiple CVEs.

We divided all fix commits into buckets based on their Prospector scores, with a step of 5. The total number of samples in each bucket can be seen in Fig. 2 (note that the Y-axis is log scaled). We can see that the population of samples with lower scores is significantly higher compared to higher scores, as there are some rules used by Prospector with low scores that match with commits more frequently, but they are less indicative of the commit relevance.

Table 2: The effect of choosing different thresholds on the size of the dataset, with and without using direct commits.

Threshold	CVEs	Projects	Commits	Accuracy	Samples
0	42,581	6,481	231,634	0.1152	395,649
35	13,872	3,242	22,016	0.8714	27,913
65	12,011	2,837	16,592	0.9588	19,865
110	3,030	1,046	3,497	1.0000	3,822
0†	57,717	11,673	247,445	0.1516	412,645
35†	29,008	8,434	37,827	0.9201	44,909
65†	27,147	8,029	32,403	0.9778	36,861
110†	18,166	6,238	19,308	1.0000	20,818

†Including direct commits mentioned in the CVE description.

For manual inspection, we used stratified random sampling and chose 500 samples for population size based on our time budget for reviewing, and randomly selected samples from each bucket based on the population of each bucket. For buckets that had fewer than 5 candidates for manual review, we took all samples. This stratified sampling resulted in a total of 712 samples to review. Then, two authors that are experienced in secure software development and vulnerability research, jointly compared the CVE descriptions and the GitHub commit information of the selected samples and assessed whether they were related. To make the assessment unbiased, reviewers did not have access to the score assigned by the Prospector. To assist in the manual review process, we developed a web application that displays the CVE description and GitHub commit page side-by-side. The review process of all 712 samples took ≈11 hours.

The results of our manual inspection can be seen in Fig. 3 (the Y-axis is log scaled). For the score range of 65-70 and higher, we can see that the majority of the inspected commits are related to their CVEs. To reduce noise in the dataset we thus chose the score of 65 as the minimum acceptable score for adding a candidate commit fix to the dataset.

The effect of choosing different thresholds on the dataset size is summarized in Table 2, with and without using direct commits that are mentioned in the CVE description. Before applying the threshold, the median of commit scores was 14, and 94.9% of commits were scored below 65. Considering the related and unrelated CVE-commit samples per population, we can estimate that if we choose 65 as the threshold, there will be approximately 4.12% noise in the dataset that arises from the Prospector heuristics. We defined a sample as noise where the fix commit is not related to its related specific CVE in the dataset. When combined with direct commit fixes mentioned in the advisory, which are related to their CVEs by default, the total noise ratio should be reduced to 2.12%.

We use only the samples with scores ≥ 65 in our dataset that we analyze and compare with others in the next section. However, we keep the data for samples with scores lower than 65 in the database for future reuse by other researchers. In our dataset collection tool, it is easy to set the threshold to customize the dataset.

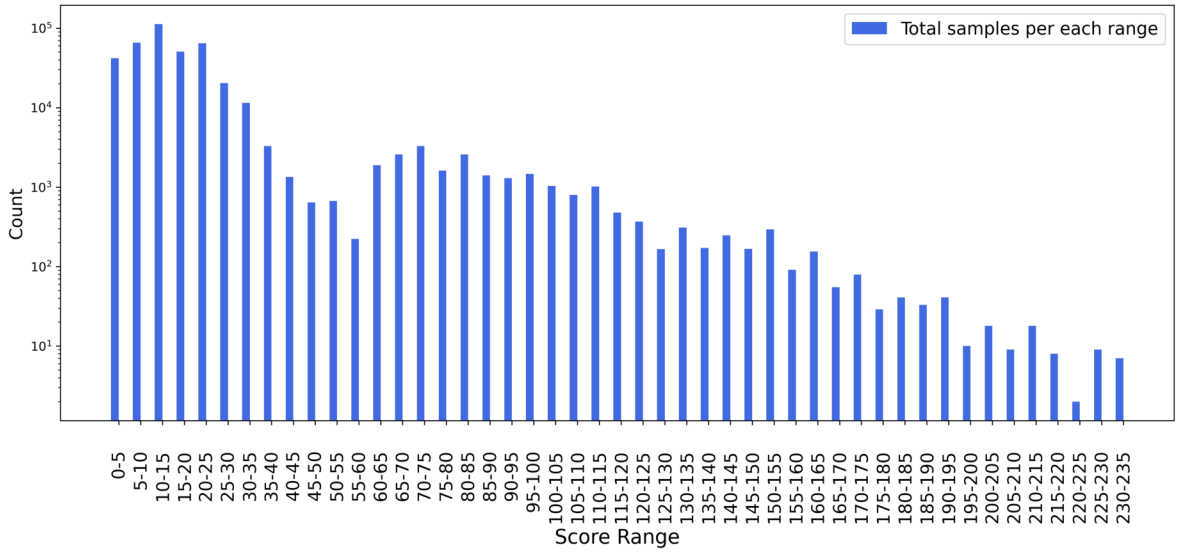

Figure 2: Total number of extracted fix commit candidates from Prospector for each bucket (log scaled).

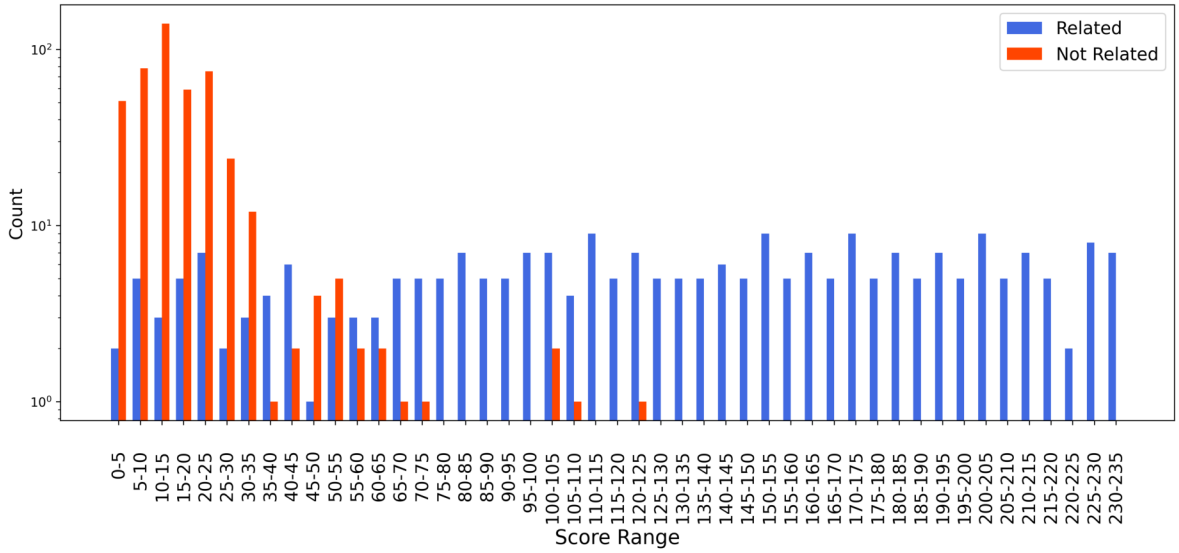

Figure 3: Manual assessment results of the relevance between discovered commits and their related CVE per each range, based on stratified random sampling (log scaled).

3 DATASET DESCRIPTION

In this section, we describe the format of our dataset that we called `MoreFixes` and compare it with datasets published in prior work.

3.1 PostgreSQL Database

Similarly to CVEFixes [4], we extract all data related to a fix commit, such as repository metadata (stars, programming language) and commit information such as changed files, changed methods, changed lines, etc. We replicate the data structure of CVEFixes and extend it in a compatible way (see Fig. 4). Thus, all prior approaches that rely on CVEFixes can be easily applied to our extended dataset.

As can be seen in the figure, two new columns that we have introduced are shown in blue. These store the commit score (from Prospector) and the type of relationship that made detecting the relation between CVE and repository possible. In the construction step a few tables were introduced that are used for the internal construction process. The most important of these is a table named `cve_project` which stores the relation of a CVE (`cve_id`) with a GitHub repository (`repo_url`), with the type of relation (`rel_type`). This table includes all CVE to repository relation samples that were not added directly to the `fixes` table. The final database is shared

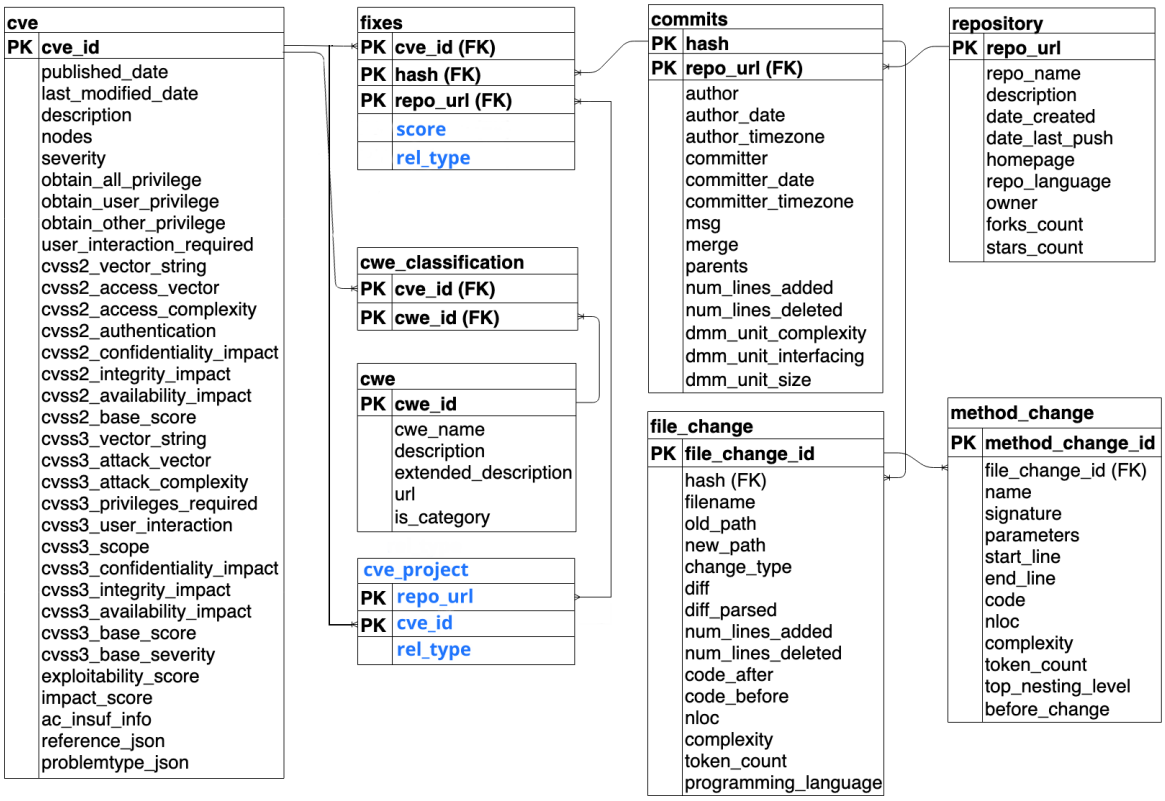

Figure 4: An entity relationship diagram of the `MoreFixes` dataset based on [4], with new columns and tables added in blue. PK stands for Primary Key; FK for Foreign Key.

with the community as a PostgreSQL database dump, including the tables used in the construction process.

Patch Files. We also publish all extracted commits as separate patch files. There are two main motivations to do so. Firstly, large commits (e.g. modifying binary files) would crash the detection process of changed methods. Having the patch files allows researchers who are only interested in the commits to extract the code from patches without having to run the full database, as it would be time-consuming to extract all patch files manually. Secondly, in some cases, the commit is not available inside the cloned repository but is visible in the GitHub web interface (e.g. a commit that is available only in a fork). It is worth mentioning that this GitHub feature can sometimes be used unethically by showing a commit outside of the repository[11]. However, in CVE descriptions this is not common and links are reviewed by CVE publishers. We used GitHub directly to extract such patches. Patch files are stored with the name "github_com_PROJECT-NAME_COMMIT-HASH" where hash and project name can be derived from the database. This can help future studies that rely on patch files as input.

3.2 Dataset Statistics

In the following `MoreFixes` reference statistics, we only include direct links to fix commits and commits detected by Prospector with a minimum score of 65. In total, `MoreFixes` covers fix commits for 26,617 CVEs from 1999 until January 24, 2024. In Fig. 5, we present the distribution of the number of CVEs and the number of CVEs we extracted with their fixes over the years (the Y-axis is log-scaled).

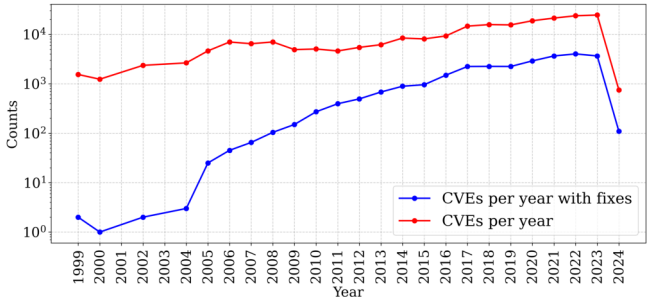

Figure 5: Distribution of CVEs and CVEs with fixes over the years.

Table 3 summarizes the top occurrences of different CWE vulnerability types. Similar to what has been disclosed for CVEFixes [4], we also detect that CWE-79 is the most repeated type of vulnerability. `MoreFixes` covers 309 different vulnerability types, which is an ≈58% increase compared to the CWEs covered by CVEFixes (180 different types). The top 10 most frequent CWE types in our dataset

[11] https://github.com/torvalds/linux/commit/b4061a10fc29010a610ff2b5b20160d7335e69bf

Table 3: Comparison of top CWE distributions over vulnerabilities in our work and CVEFixes.

CWE	Description	CVEs	
		CVEFixes	MoreFixes
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	635	**3,479**
Noinfo†	Insufficient Information	193	**1,479**
CWE-787	Out-of-bounds Write	205	**1,471**
CWE-125	Out-of-bounds Read	380	**1,333**
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	408	**1,149**
CWE-20	Improper Input Validation	382	**1,119**
CWE-476	NULL Pointer Dereference	195	**994**
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	130	**795**
Other*	Other	165	**792**
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	276	**686**

†NVD-CVE-noinfo
* NVD-CWE-other

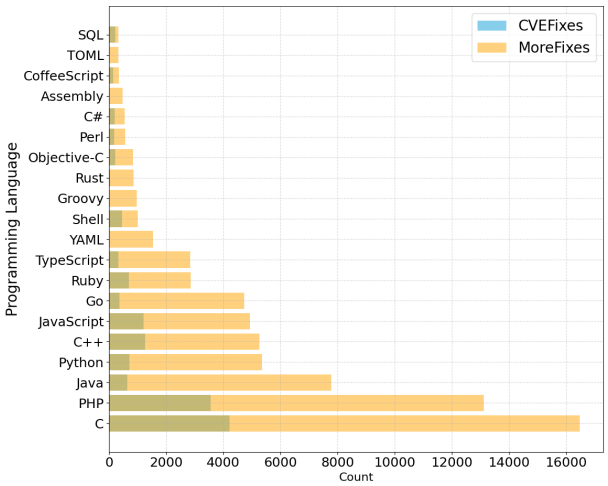

Figure 6: Distribution of fixes in different programming languages, compared to CVEFixes [4].

are similar to CVEFixes, except for having CWE-89 (SQL Injection) in our top 10, while CVEFixes had instead CWE-264 (Permissions Privileges and Access Control).

In `MoreFixes`, we identified 54 different file types related to programming languages and environment configurations, which is significantly higher compared to the identified file formats in CVEFixes (31). Our dataset includes programming languages that are not available in CVEFixes, such as Dart, Kotlin, Groovy, Pascal, etc. Fig. 6 presents a distribution of the number of file changes related to fixes per programming language in our dataset compared to CVEFixes, in top-repeated samples of programming languages.

Note that sometimes a fix commit contains both data related to a fix of code (i.e. the patch) and documentation. For example, a commit in the Linux kernel[12], that fixed CVE-2013-4312, changed a text file for documentation and the relevant C code with the same commit. Thus, there are fix commits with different file changes that can be selected in future studies with diverse objectives, such as understanding vulnerable code changes or studying the changes documentation.

Table 4 presents the top 10 projects in `MoreFixes` with the most CVEs and the boxplots of the Common Vulnerability Scoring System (CVSS) scores of these CVEs. Usually, vulnerabilities with very low severity scores are not assigned a CVE. This can also be seen in the CVSS column of Table 4: the minimum CVSS score of CVEs is greater than 2 and most of the scores are higher than 4, which is the minimum threshold for medium impact severity.

Table 4: Top repositories with most CVEs

Repository	CVEs	CVSSv3 Score
Linux	2,984	
ImageMagick	600	
Wireshark	495	
Gpac	479	
Tensorflow	473	
Qemu	380	
Php-src	313	
FFmpeg	307	
Moodle	274	
Go	262	

3.3 Datasets Comparison

To make a fair comparison between our dataset and prior work, we chose the top 5 projects (Linux, Tcpdump, phpMyAdmin, ImageMagick and Tensorflow) with the most CVEs in CVEFixes[13] [4], CrossVul[14] [27], Reis and Abreu[15] [32] and `MoreFixes`, and only considered the years 2010 until 2020, as the other works were published in 2021. The visualization of the dataset intersections in Fig. 7 shows that most of the CVEs from CVEFixes, dataset by Reis and Abreu [32] and CrossVul are shared. Although we did not merge their data dumps directly in our work, it can be seen that almost all CVEs (except two[16]) detected in prior work are also detected by our workflow. Furthermore, our dataset contains 821 unique CVEs

[12] https://github.com/torvalds/linux/commit/759c01142a5d0f364a462346168a56de28a80f52
[13] https://zenodo.org/records/7029359
[14] https://zenodo.org/records/4734050
[15] https://github.com/TQRG/security-patches-dataset

[16] Our heuristics did detect the correct GitHub links for these 2 CVEs, but the fix commit detection (step F) failed.

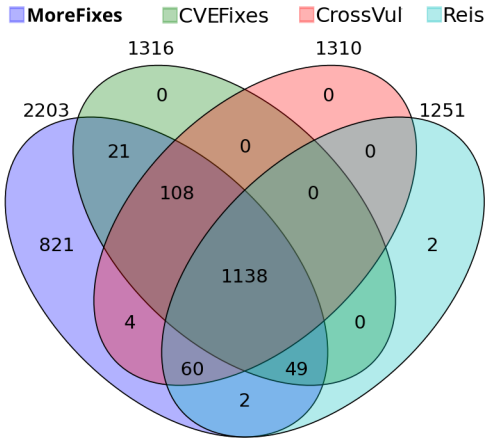

Figure 7: Comparison between `MoreFixes`, CrossVul [27], Reis and Abreu [32] and CVEFixes [4] for unique CVEs with commit fixes in top 5 projects with most CVEs included in the datasets.

Table 5: Comparison of `MoreFixes` to prior work.

Dataset	CVEs	Projects	Commits	CVE Years
DiverseVul [7]	N/A	797	7,514	1999-2023
DiverseVul (mix)† [7]	N/A	933	21,949	1999-2023
ProjectKB* [18]	624	205	1,282	2007-2019
BigVul* [12]	3,754	348	4,432	2002-2019
CrossVul [27]	5,138	1,675	5,877	1999-2021
CVEFixes [4]	5,365	1,754	5,495	1999-2021
Reis (mix)† [32]	5,942	1,339	8,057	2002-2019
`MoreFixes`	**26,617**	**6,945**	**31,883**	**1999-2024**

* Contains only one programming language
†Including merged results of other works

(60% increase) with fix commits that were not previously detected in the same projects and date range.

Table 5 presents an overall comparison between `MoreFixes` and prior work. As of January 24, 2024, our dataset contains 26,617 unique CVEs coming from 6,945 unique GitHub projects, which is a 397% increase in the number of CVEs and a 295% increase in included open-source projects compared to CVEFixes. Concerning these CVEs, `MoreFixes` comprises 31,883 unique commits that fixed the vulnerabilities, which is a 480% increase compared to CVEFixes. Note that we did not explicitly focus on any specific set of programming languages.

3.4 Impact of Our Heuristics

Table 6 helps to assess the contribution of our new heuristics used to extract fix commits. It can be seen that ≈53% of the final commits available in our dataset are direct results of our novel methodology (steps B, C, D). Notably, we see that the GitHub Security Advisory (GHSA) shows promising results in gathering direct commit fixes and CVE to open-source project mappings. It is worth mentioning that whenever a direct fix commit or a CVE to repository address

Table 6: Distribution of CVE to repository mapping with different heuristics using their reference step identifiers as given in Fig. 1.

Relation type	Occurrences
NVD direct commit (Step **A**)	13,736
NVD repository based (Step **D**)	9,284
CPE repository based (Step **B**)	7,234
GHSA direct commit (Step **A**)	3,260
CPE direct commit (Step **B**)	2,191
GHSA repository based (Step **D**)	531
GitHub API search (Step **B**)	374
GHSA registry address (Step **C**)	251

mapping was identified, no duplicate records were inserted in the database. Thus, it is not possible to compare the quality of the NVD dataset with GHSA based on the presented table. Early termination of the workflow is also another reason why Step C (the last step in the flowchart) in our pipeline brought so few results.

4 LIMITATIONS

In this section, we discuss the limitations of our work. As we build on the state-of-the-art methods like CVEFixes [4] and Prospector [34], we also inherit many of their limitations. Notably, we have updated the CVEFixes workflow and its key limitation of starting from scratch every time (see Sec. 5.2 in [4]) is not applicable to our tool, as we made the process incremental. Other CVEFixes limitations, such as relying on the stability of NVD and other databases, using only GitHub sources, and not considering the project license information, are still applicable to our work. For example, many different public or self-hosted services include open-source software, such as Chrome, Android, and several self-hosted Git-based services. By adding such services, it is possible to further increase the quantity of samples in this dataset.

Information related to CVEs (such as vulnerability description, or version tagging) or CWEs is sometimes labeled incorrectly or missing in the source databases like NVD, and so we inherit this limitation. For example, as mentioned in Table 3, there are 2,271 samples without CWE types. By adding further heuristic or machine learning methods it should be possible to improve data labeling.

The overall limitation of the repository mining approaches is the availability of data. Repositories can be removed or renamed, and thus the corresponding fix commits or patch files will not be retrievable. This explains the slight discrepancy in the numbers in Tables 2 and 5. Table 5 shows the final size of the `MoreFixes` dataset, which comprises the retrieved fix commits and patch files.

Our heuristics, for example, the blocklist keywords used to ignore non-related GitHub repositories, might have introduced some noise into the dataset. Also, noise may appear in our CVE to repository mapping step. However, in the manual review step, we did not detect any incorrect CVE to repository linking for scores above 65. This might be due to the Prospector's rules that prevent the commits in the wrong repository from getting a high score.

Still, as we have shown, the Prospector heuristics generate some noise and select commits that occasionally might not be relevant

to the target CVEs. We have tried to reduce this noise as much as possible by evaluating it in a manual review and selecting a reasonable Prospector score threshold, but the resulting dataset might still include some irrelevant commits. We share the Prospector scores so that it is possible select a more reliable commit subset. We note that most of the high-score yet irrelevant commits were security fixes, but not related to assigned CVE. Another option for future improvements in detecting fix commits is to combine Prospector with advanced generative AI-based methods like VCFinder [10].

Moreover, in fix commits, sometimes non-security-related changes are mixed in the same commit with security-related changes. For example, test cases are often implemented to verify fixing the vulnerability. Such changes might be useful in specific research, but such data must be ignored for studies like automated vulnerability detection. An additional data-cleaning approach is required to prune such non-security changes. This data quality issue was also studied by Croft et al. [9]. As mentioned in [9], some existing datasets are noisy due to poor keyword choices. We tackled this issue by using Prospector as a more robust approach. Moreover, in our dataset, each vulnerability fix commit is assigned to at least one CVE, which allows proper tracing and validation. With a custom threshold setting for vulnerability score or limiting top score commits per CVE, it is possible to further increase the overall accuracy of available commits, at the cost of a reduced dataset size.

5 RELATED WORK

In this section, we discuss the related literature on vulnerability datasets and their common applications.

Existing Datasets. As we mentioned, the majority of existing vulnerability datasets are either generated or mined through publicly available open-source software projects. Generated vulnerability datasets are usually less used due to their artificial nature and lack of similarity with real-world datasets. CVEFixes [4], on which we build, uses the NVD CVEs data and attempts to detect commit links in the CVE record references. Subsequently, all data related to CWEs, CVEs, repository, code, and file changes are stored in a database. The CrossVul [27] dataset was collected using a similar approach, but by only gathering commits as simple files, making it harder to execute SQL queries for complex data joining and analysis. DiverseVul [7] implemented their database extraction by choosing a different vulnerability source and limiting it to C/C++. However, one of the main data sources of their work is no longer publicly available, preventing it from having an updated version of the data. BigVul [12] also focused on the C/C++ programming languages and it relies on a limited set of known high-quality projects such as Chrome, Android, and Linux. Ponta et al. [30] manually created a dataset for the Java programming language by mining vulnerability disclosure information from various security websites such as NVD. Similarly, Reis et al. [32] created a dataset by mining CVE details and augmenting their dataset with other security datasets. In a recent research, Wang et al. [41] created the ReposVul dataset by choosing a vulnerability website that references GitHub, and two projects from Google, covering only four programming languages.

Vulnerability Research and Its Applications. Vulnerability datasets commit fixes have various applications such as automatic vulnerability repairing with machine learning [8, 13, 14, 16, 19], automatic bug generation [20, 28, 29], vulnerability detection [15, 31], benchmarking security analysis tools [21, 43], understanding and studying security commits [33, 38], vulnerability fix commits identification[40]. Quality and quantity of the data from vulnerability datasets and the diversity of projects and languages covered by them are important aspects that `MoreFixes` addresses.

6 CONCLUSIONS

We have shown that by using several smart heuristics and combining state-of-the-art approaches it is possible to substantially improve the discovery of CVE fix commits in open-source projects and, as a byproduct, increase the number of different types of discovered vulnerabilities. Thanks to our approach, we publish the largest programming language-independent real-world CVE vulnerability dataset with fixes. Compared to the state-of-the-art methods, our dataset also includes significantly more samples, patch files, CWEs, and the relation between a public CVE and the corresponding GitHub repository related to the CVE. Our `MoreFixes` dataset can be used by the community to study vulnerabilities in OSS, vulnerability detection, common vulnerable patterns, and static application security testing benchmarks.

DECLARATIONS

Data-Availability Statement. Dataset [2] and source code [3] are available on Zenodo.

Funding Statement. This research was partially supported by the Dutch Research Council (NWO) under the project NWA.1215.18.008 Cyber Security by Integrated Design (C-SIDe).

REFERENCES

[1] 2022. Machine Learning for Source Code Vulnerability Detection: What Works and What Isn't There Yet. *IEEE Security & Privacy* 20, 5 (2022), 60–76.
[2] Jafar Akhoundali, Sajad Rahim Nouri, Kristian F. D. Rietveld, and Olga GADYATSKAYA. 2024. *MoreFixes: Largest CVE dataset with fixes.* https://doi.org/10.5281/zenodo.11199120
[3] Jafar Akhoundali, Sajad Rahim Nouri, Kristian F. D. Rietveld, and Olga GADYATSKAYA. 2024. *Source code for "MoreFixes".* https://doi.org/10.5281/zenodo.11110595
[4] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEFixes: Automated collection of vulnerabilities and their fixes from open-source software. In *Proc. of PROMISE*. 30–39.
[5] Tim Boland and Paul E Black. 2012. Juliet 1. 1 C/C++ and Java test suite. *Computer* 45, 10 (2012), 88–90.
[6] Nicholas Chan and John A Chandy. 2022. Extracting vulnerabilities from GitHub commits. In *Proc. of SANER*. IEEE, 235–239.
[7] Yizheng Chen, Zhoujie Ding, Xinyun Chen, and David Wagner. 2023. DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. *arXiv preprint arXiv:2304.00409* (2023).
[8] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2022. Neural transfer learning for repairing security vulnerabilities in C code. *IEEE Transactions on Software Engineering* 49, 1 (2022), 147–165.
[9] Roland Croft, M Ali Babar, and M Mehdi Kholoosi. 2023. Data quality for software vulnerability datasets. In *Proc. of ICSE*. IEEE, 121–133.
[10] Trevor Dunlap, Elizabeth Lin, William Enck, and Bradley Reaves. 2023. VFCFinder: Seamlessly pairing security advisories and patches. *arXiv preprint arXiv:2311.01532* (2023).
[11] Soufian El Yadmani, Robin The, and Olga Gadyatskaya. 2022. Beyond the Surface: Investigating Malicious CVE Proof of Concept Exploits on GitHub. *arXiv preprint arXiv:2210.08374* (2022).
[12] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. AC/C++ code vulnerability dataset with code changes and CVE summaries. In *Proc. of MSR*. 508–512.
[13] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Yuki Kume, Van Nguyen, Dinh Phung, and John Grundy. 2024. AIBughunter: A practical tool for predicting, classifying and repairing software vulnerabilities. *Empirical Software Engineering* 29, 1 (2024), 4.

[14] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. 2022. VulRepair: A T5-based automated software vulnerability repair. In *Proc. of ESEC/FSE*. 935–947.

[15] Michael Fu, Chakkrit Tantithamthavorn, Van Nguyen, and Trung Le. 2023. ChatGPT for vulnerability detection, classification, and repair: How far are we? *arXiv preprint arXiv:2310.09810* (2023).

[16] Anastasiia Grishina. 2022. Enabling automatic repair of source code vulnerabilities using data-driven methods. In *Proc. of ICSE: Companion Proceedings*. 275–277.

[17] Yuejun Guo and Seifeddine Bettaieb. 2023. An Investigation of Quality Issues in Vulnerability Detection Datasets. In *Proc. of EuroS&P Workshops*. IEEE, 29–33.

[18] Daan Hommersom, Antonino Sabetta, Bonaventura Coppola, Dario Di Nucci, and Damian A. Tamburri. 2021. Automated Mapping of Vulnerability Advisories onto their Fix Commits in Open Source Repositories. https://arxiv.org/pdf/2103.13375.pdf

[19] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. 2023. An empirical study on fine-tuning large language models of code for automated program repair. In *Proc. of ASE*. IEEE, 1162–1174.

[20] Ali Reza Ibrahimzada, Yang Chen, Ryan Rong, and Reyhaneh Jabbarvand. 2023. Automated Bug Generation in the era of Large Language Models. *arXiv preprint arXiv:2310.02407* (2023).

[21] Avishree Khare, Saikat Dutta, Ziyang Li, Alaia Solko-Breslin, Rajeev Alur, and Mayur Naik. 2023. Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities. *arXiv preprint arXiv:2311.16169* (2023).

[22] Maryna Kluban, Mohammad Mannan, and Amr Youssef. 2023. On Detecting and Measuring Exploitable JavaScript Functions in Real-World Applications. *ACM TOPS* (2023).

[23] Arina Kudriavtseva and Olga Gadyatskaya. 2024. You cannot improve what you do not measure: A triangulation study of software security metrics. In *Proc. of SAC*. ACM.

[24] Frank Li and Vern Paxson. 2017. A large-scale empirical study of security patches. In *Proc. of CCS*. 2201–2215.

[25] Truong Giang Nguyen, Thanh Le-Cong, Hong Jin Kang, Xuan-Bach D Le, and David Lo. 2022. Vulcurator: a vulnerability-fixing commit detector. In *Proc. of ESEC/FSE*. 1726–1730.

[26] Giang Nguyen-Truong, Hong Jin Kang, David Lo, Abhishek Sharma, Andrew E Santosa, Asankhaya Sharma, and Ming Yi Ang. 2022. Hermes: Using commit-issue linking to detect vulnerability-fixing commits. In *Proc. of SANER*. IEEE, 51–62.

[27] Georgios Nikitopoulos, Konstantina Dritsa, Panos Louridas, and Dimitris Mitropoulos. 2021. CrossVul: A cross-language vulnerability dataset with commit data. In *Proc. of ESEC/FSE*. 1565–1569.

[28] Yu Nong, Richard Fang, Guangbei Yi, Kunsong Zhao, Xiapu Luo, Feng Chen, and Haipeng Cai. 2023. VGX: Large-scale Sample Generation for Boosting Learning-Based Software Vulnerability Analyses. *arXiv preprint arXiv:2310.15436* (2023).

[29] Yu Nong, Yuzhe Ou, Michael Pradel, Feng Chen, and Haipeng Cai. 2023. VULGEN: Realistic Vulnerability Generation Via Pattern Mining and Deep Learning. In *Proc. of ICSE*.

[30] 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 *Proc. of MSR*. IEEE, 383–387.

[31] Moumita Das Purba, Arpita Ghosh, Benjamin J Radford, and Bill Chu. 2023. Software vulnerability detection using large language models. In *Proc. of ISSRE Workshops*. IEEE, 112–119.

[32] Sofia Reis and Rui Abreu. 2021. A ground-truth dataset of real security patches. *arXiv preprint arXiv:2110.09635* (2021).

[33] Sofia Reis, Rui Abreu, Hakan Erdogmus, and Corina Păsăreanu. 2022. SECOM: Towards a convention for security commit messages. In *Proc. of MSR*. 764–765.

[34] Antonino Sabetta, Serena Elisa Ponta, Rocio Cabrera Lozoya, Michele Bezzi, Tommaso Sacchetti, Matteo Greco, Gergő Balogh, Péter Hegedűs, Rudolf Ferenc, Ranindya Paramitha, et al. 2024. Known Vulnerabilities of Open Source Projects: Where Are the Fixes? *IEEE Security & Privacy* (2024).

[35] Felix Schuckert, Basel Katt, and Hanno Langweg. 2023. Insecurity Refactoring: Automated Injection of Vulnerabilities in Source Code. *Computers & Security* 128 (2023), 103121.

[36] K Sivakumar and K Garg. 2007. Constructing a "common cross-site scripting vulnerabilities enumeration (CXE)" using CWE and CVE. In *Proc. of ICISS*. Springer, 277–291.

[37] Bertrand Stivalet and Elizabeth Fong. 2016. Large scale generation of complex and faulty PHP test cases. In *Proc. of ICST*. IEEE, 409–415.

[38] Shiyu Sun, Shu Wang, Xinda Wang, Yunlong Xing, Elisa Zhang, and Kun Sun. 2023. Exploring Security Commits in Python. In *Proc. of ICSME*. IEEE, 171–181.

[39] Norbert Tihanyi, Tamas Bisztray, Ridhi Jain, Mohamed Amine Ferrag, Lucas C Cordeiro, and Vasileios Mavroeidis. 2023. The FormAI dataset: Generative AI in software security through the lens of formal verification. In *Proc. of PROMISE*. 33–43.

[40] Hieu Dinh Vo, Thanh Trong Vu, and Son Nguyen. 2023. Silent Vulnerability-fixing Commit Identification Based on Graph Neural Networks. *arXiv preprint arXiv:2309.08225* (2023).

[41] Xinchen Wang, Ruida Hu, Cuiyun Gao, Xin-Cheng Wen, Yujia Chen, and Qing Liao. 2024. A Repository-Level Dataset For Detecting, Classifying and Repairing Software Vulnerabilities. *arXiv preprint arXiv:2401.13169* (2024).

[42] Xinda Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. 2019. Detecting" 0-day" vulnerability: An empirical study of secret security patch in OSS. In *Proc. of DSN*. IEEE, 485–492.

[43] Yi Wu, Nan Jiang, Hung Viet Pham, Thibaud Lutellier, Jordan Davis, Lin Tan, Petr Babkin, and Sameena Shah. 2023. How Effective Are Neural Networks for Fixing Security Vulnerabilities. *arXiv preprint arXiv:2305.18607* (2023).

Received 2024-03-28; accepted 2024-04-19