MegaVul: A C/C++ Vulnerability Dataset with Comprehensive Code Representations

Chao Ni, Liyu Shen, Xiaohu Yang, Yan Zhu*
{chaoni,liyushen,yangxh,yan.zhu}@zju.edu.cn
Zhejiang University
Hangzhou, China

Shaohua Wang
davidshwang@ieee.org
Central University of Finance and Economics
Beijing, China

ABSTRACT

We constructed a newly large-scale and comprehensive C/C++ vulnerability dataset named **MegaVul** by crawling the Common Vulnerabilities and Exposures (CVE) database and CVE-related open-source projects. Specifically, we collected all crawlable descriptive information of the vulnerabilities from the CVE database and extracted all vulnerability-related code changes from 28 Git-based websites. We adopt advanced tools to ensure the extracted code integrality and enrich the code with four different transformed representations. Totally, MegaVul contains 17,380 vulnerabilities collected from 992 open-source repositories spanning 169 different vulnerability types disclosed from January 2006 to October 2023. Thus, MegaVul can be used for a variety of software security-related tasks including detecting vulnerabilities and assessing vulnerability severity. All information is stored in the JSON format for easy usage. MegaVul is publicly available on GitHub and will be continuously updated. It can be easily extended to other programming languages.

CCS CONCEPTS

• **Software and its engineering → Software defect analysis**.

KEYWORDS

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

ACM Reference Format:
Chao Ni, Liyu Shen, Xiaohu Yang, Yan Zhu and Shaohua Wang. 2024. MegaVul: A C/C++ Vulnerability Dataset with Comprehensive Code Representations. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643991.3644886

1 INTRODUCTION

Data-driven software vulnerability analysis has been the core and critical activity in both industry and academia. In particular, vulnerability detection attracts much attention from the research community [12, 17–19] since undetected vulnerabilities can be exploited by hackers and consequently lead to great losses to users. For example, a newly exploited vulnerability in a commonly used logging tool *Apache Log4J* (CVE-2021-44228) makes millions of applications that adopt Log4J for logging execution information under attack. Therefore, many studies proposed several datasets for better detecting software vulnerabilities. However, existing datasets still have some limitations that may impact the verification of proposed models, including ① *unreal vulnerability* (i.e., SARD [1] is artificially synthesized), ② *unreal data distribution* (i.e., balanced distribution in Devign [20]), ③ *limited diversity* (i.e., limited projects and vulnerability types in ReVeal [9]), ④ *limited newly disclosed vulnerabilities* (i.e., no updating to Big-Vul [10] covering the period only from 2003 to 2019), and ⑤ *low-quality of dataset* (i.e., incomplete function, erroneously merged functions, missed commit message in Big-Vul [10]).

Considering the aforementioned limitations, we constructed a new high-quality, data-rich, multi-dimensional C/C++ vulnerability dataset named **MegaVul** from the Common Vulnerabilities and Exposures (CVE) database [2] 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 severity score, references linking to the affected products, etc). Second, through the CVE references directly or indirectly, we dug into the relevant products hosted on Git-based websites (i.e., 28 in total) and then identified these vulnerability-related code commits and extracted relevant information (e.g., metadata, commit files). We adopt advanced tools to ensure the extracted code integrality and to enrich the dimension of information. In total, MegaVul contains 17,380 vulnerabilities collected from 992 open-source repositories spanning 169 different vulnerability types disclosed from January 2006 to October 2023. Notice that the scripts for collecting MegaVul are publicly available and can be easily extended to other programming languages and implemented with more functionality. MegaVul can be used for a variety of software security-related tasks, but not limited to, ① deep analysis on vulnerability characteristics and ② data-driven vulnerability detection and identification of vulnerability fixing patches. Eventually, the contributions of our paper are summarized as follows:

[A. Continuously updating Dataset.] We collected and published a large-scale dataset with comprehensive basic information from both the CVE dataset and open-source project repositories as well as the enriched transformed representation of involved codes.

[B. Open-soured Collection Process.]The data collection approach with supporting scripts is publicly available on GitHub [5].

*Both Chao Ni and Liyu Shen contribute equally.
Yan Zhu is the corresponding author.

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04
https://doi.org/10.1145/3643991.3644886

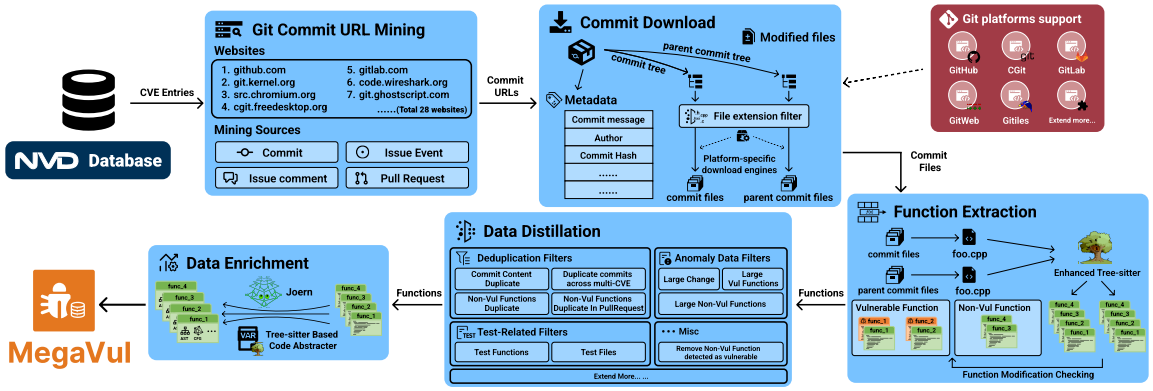

Figure 1: MegaVul data collection pipeline

2 DATASET CONSTRUCTION

To address the aforementioned limitations in the previous dataset, we crawl data from more open-source repositories, using sophisticated filtering methods to improve the quality of the collected data. Furthermore, we employ advanced techniques to extract functions. Apart from collecting the raw functions, we also provide additional dimensions information of functions, including the abstracted versions and the information in the form of graphs obtained by *Joern* [4] such as the AST and the CFG of the functions. This supplementary information avoids the re-implementation for those people who want to extract this information since it requires a relatively complex setup and consequently facilitates further research in this field. Table 1 presents the comparative statistical information between our dataset MegaVul and Big-Vul. Meanwhile, Figure 1 depicts the workflow for data collection with five steps, and each step of the pipeline will be described in detail as follows.

Table 1: Descriptive statistics for MegaVul and Big-Vul

	MegaVul	Big-Vul
Number of Repositories	**992**	310
Number of CVE IDs	**8,254**	3,539
Date range of crawled CVEs	**2006/01~2023/10**	2013/01~2019/03
Number of CWE IDs	**169**	92
Number of Commits	**9,019**	4,058
Number of Crawled Websites	**28**	2
Number of Vul/Non-Vul Function	**17,380/322,168**	10,900/177,736
Function Extract Method/(Quality)	**Tree-sitter/(High)**	Lizard/(Low)
Dimensions of Information	**6**	2
Code Integrality	**Full**	Partial

❶ Git Commit URL Mining. The National Vulnerability Database (NVD) [6] is a public resource maintained by NIST to collect, organize, and distribute information on security vulnerabilities in software products, which is a good starting point for collecting a high-quality, comprehensive dataset for vulnerability detection. We crawl all available Common Vulnerabilities and Exposures (CVE) [2] entries provided by the NVD into JSON format, which contains valuable descriptive information, including *CVE summary*, *CVSS scores*, and *reference links to relevant products* or *patches for fixing the vulnerabilities*. However, the majority of references do not directly link to the URL of the software's commit but to the reference security bulletin or vulnerability advisory. As a result, we conducted a manual analysis of the referenced commit links in CVE entries, identifying 28 Git-based code hosting platforms from 368 websites that had referenced the CVE entries more than 50 times. Apart from identifying directed commit URLs, we also employed three other methods to mine potential commit URLs from the candidate websites: ① *Issue Event*: Locating the *ClosedEvent* or *CrossReferencedEvent* in the issue timeline that references commit URL; ② *Issue Comment*: Extracting the commit URL from specific comment formats within issues, such as the commit message comment autogenerated by the BOT; ③ *Pull Request (PR)*: Identifying direct PR URL and mine PR URL from issues;

❷ Commit Download. In the previous step, the obtained commit URLs point to vulnerability-fixing commits. Following previous work [9], for each modified function in a vulnerability fix commit, the previous version of the function in the commit (i.e., the parent commit) is annotated as vulnerable, while the current version of the function is treated as non-vulnerable. Meanwhile, those functions that remain unchanged are also annotated as non-vulnerable. Therefore, in this step, we download the metadata of commits as well as the files that have been modified compared to the parent commit. To minimize the size of downloading irrelevant data, we only gather C/C++-related source codes and headers based on their file extensions. These commit URLs are sourced from various Git web hosting services, and to download commits from these platforms, we categorize these code hosting platforms into five main categories: GitHub, GitLab, GitWeb, CGit, and Gitiles. For each of these platforms, we implement the corresponding scripts to download commits. For example, we utilize the official GitHub API to handle downloads from GitHub and employ a crawler for GitWeb to traverse the commit's metadata and extract the download path of the files.

❸ Function Extraction. Separating the functions in each file is a crucial step to ensure the quality and integrality of function extraction. Existing datasets [1, 7, 10, 16, 20] are collected through predefined rules or using regular expression tools like *lizard*. However, these methods have several limitations in their functionalities due to their simplicity. For example, regular expressions cannot fully represent the complex syntax rules of the C-like programming language. As a result, these methods may result in the extraction of incomplete functions or the incorrect separation of multiple functions into a single function. Oppositely, we adopt a complex, syntax rule-based parser *tree-sitter* to separate functions, and enhance the syntax rules to identify more macro modifiers frequently used by popular open-source repositories, such as the macro modifier `__init` defined in Linux. Furthermore, we temporarily remove the macros from the code before parsing, which allows the parser

to effectively and accurately separate the functions without the interference of macros.

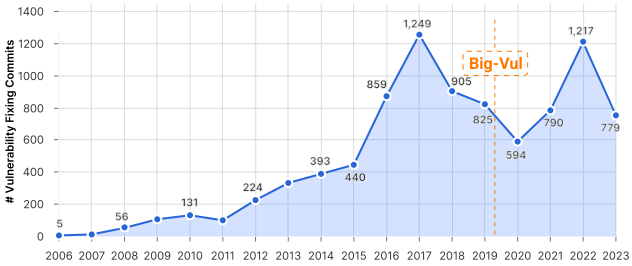

Figure 2: Commit Count of Annual Vulnerability

❹ Data Distillation. To address the noise in the raw dataset, multiple filters are applied to obtain the high-quality dataset, reducing data redundancy and leakage, and ensuring that the model trained with MegaVul demonstrates high performance, fairness, and reliability. ① *Deduplication Filters* aim to filter out duplicated functions to keep the dataset simple. When a commit fixes multiple vulnerabilities and is referenced by multiple CVEs, we keep only one commit copy to ensure a single data source. Deduplicating fixing commits with different *hash* (i.e., an ID to indicate a commit) might be the same since they refer to the patches on different branches. Deduplicate the same non-vulnerable functions across datasets. Deduplicate identical non-vulnerable functions in multiple commits within a single pull request. ② *Anomaly Data Filters*: To filter these templated codes generated by tools (i.e., *yacc-generated* or *antlr-generated*), we adopt a 2-sigma criterion (i.e., 95%) based on their length to filter both vulnerable or non-vulnerable functions. We also remove large-scale changes including PR with many commits since a large number of modified files or diff lines usually arise from software major version bumps or code refactoring. ③ *Test-Related Filters* aim to filter out files or functions related to unit testing based on their names. ④ *Other Filters* aim to filter non-vulnerable functions that have been annotated as vulnerable after a specific commit to ensure label consistency and prevent the same function from being annotated as both vulnerable and non-vulnerable;

❺ Data Enrichment. In this step, we aim to enrich MegaVul information by providing more types of representation. For example, graph representation is tedious work but extremely required by graph-based models [9, 14, 20]. "Abstraction" (e.g., substituting variable names with symbolic names like "VAR" [15]) is also needed by token-based sequence models to address the problem of vocabulary explosion. Eventually, MegaVul provides information on the four dimensions of the function: ① *Function Signature*: includes the function name, a comprehensive list of arguments with their respective names and types, and the return type of the function. ② *Abstracted Function*: provides nine types of granularity abstraction (e.g., FUNC, VAR, STRING, COMMENT, etc.). ③ *Parsed Function*: means the functions with several types of representations (i.e., Abstract Syntax Trees (AST), Program Dependence Graphs (PDG) [11], etc.) statically parsed by *Joern*. ④ *Code Changes*: include the details of function changes generated by *diff* tool [3]. For example, line-level modifications, additions, and deletions.

3 DATA ANALYSIS

Figure 2 clearly shows the trend in the number of C/C++-related vulnerability fixing commits per year since 2006. Overall, the number has increased dramatically since 2015, which indicates the aggressive vulnerability-fixing efforts by vendors and developers. The peak occurred in 2017 with 1,255 commits in total. In particular, *ImageMagick* project contributes the most, and the majority of vulnerability types are related to memory overflow vulnerabilities in different image formats, which led to a denial-of-service attack on remote servers and reported 357 CVEs in that year. Despite a downward trend in subsequent years, the number of vulnerability fixing commits remains relatively high-level and arrive at another peak in 2022. On the one hand, *Vim* project reported 115 CVEs of buffer overflow vulnerabilities, which is much higher than the average number reported every year (i.e., about 10 CVEs). On the other hand, the rise of deep learning has led to an increasing number of researchers who tend to use the TensorFlow framework developed with C++ backend operators to build deep learning models. Since 2021, TensorFlow has reported an average of 130 CVEs per year. Notice that the most comprehensive dataset Big-Vul stopped collecting data in March 2019, but there are still about 4,000 commits related to vulnerability fixes between 2019 and 2023. Therefore, a continuously updating dataset is very beneficial for vulnerability detection tasks, which helps to richer the number and types of vulnerabilities and to learn deep learning-based detection models.

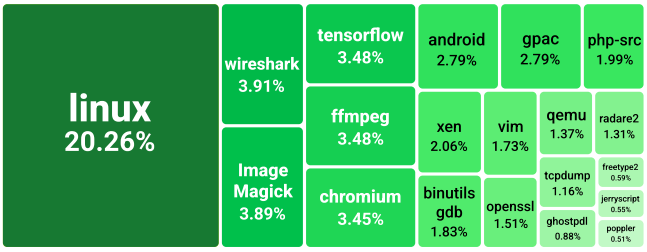

Figure 3: Distribution of the Top 20 Repositories in MegaVul

Figure 3 shows the Top-20 repositories with the highest number of CVEs in the MegaVul, which account for 59.5% of the number of CVEs recorded by MegaVul, i.e., 4,914 CVE entries. Especially, CVEs from *Linux* account for 20.26%, i.e., 1,672 CVEs. On average, MegaVul collects 110 CVEs and their corresponding vulnerability fixing commits from *Linux* every year. In addition to *Linux*, repositories such as *Wireshark*, *ImageMagick*, *tensorflow*, *ffmpeg*, and *chromium* also show a relatively high percentage, with each repository being able to collect about 300 relevant CVEs. Different from Linux which has a stable and substantial trend in the annual amount of CVEs report, other repositories may intermittently report a significant number of CVEs in particular years.

Figure 4 illustrates the Top-10 CWE types in MegaVul as well as the Top-5 repositories with the highest frequency of each CWE type. In terms of the number of CWE types, CWEs related to memory management safety dominate (i.e., CWE-119, CWE-125, CWE-787, CWE-476, CWE-20, CWE-416, and CWE-190) account for 59.27% of the total number of CWEs in MegaVul. When analyzing each CWE type, we find that the *Linux* has the highest number of vulnerabilities across multiple CWE types, except for CWE-125, which also highlights the diversity of vulnerabilities in the *Linux*. Meanwhile, we find that Top-5 frequently occurring repositories have a

wide distribution of vulnerability types which inspires us to pay attention to each category of vulnerabilities

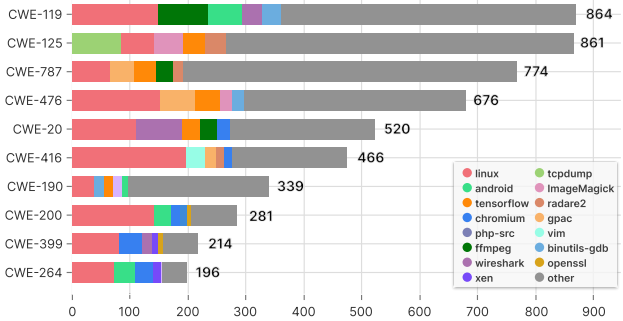

Figure 4: The Top-10 CWEs and the Top-5 repositories with most frequent occurrences in each CWE

4 DATA APPLICATION

MegaVul can be used for a variety of software safety-related tasks because of its richness of information as well as its large-scale size. We encourage researchers to use MegaVul as a benchmark to ensure a fair and uniform evaluation of SOTA performance and to avoid biased results caused by switching between different datasets.

The deep analysis on CVEs and code changes. MegaVul contains various types of textual data, such as CVE descriptions, commit messages, and code changes. By leveraging advanced NLP techniques (e.g., CodeBERT [17]), we can analyze the correlations among these textual data, understand the connotations of the code changes, identify the key information inside them, and dig deeper into the patterns of vulnerability. This analysis not only helps to determine the location of the vulnerability and assess the possible impact of the vulnerability but also provides assistance in automated code review.

Data-driven Vulnerability Detection. MegaVul covers a rich set of function information, including vulnerability functions, abstracted functions, function graphs, and other descriptive information such as the types of CWEs and CVE description. Researchers can utilize the rich information to train state-of-the-art deep learning models designed for vulnerability detection, including sequence-based and graph-based. In addition, MegaVul is collected from several real-world repositories, covering functions with and without vulnerabilities with a realistic proportion. Therefore, it supports training vulnerability detection models for better use in real-world production environments.

Identification of vulnerability fixing patches. Due to the delay in disclosing vulnerability information after they have been patched, it creates a window of opportunity for attackers. Thus, many methods have been proposed for the identification of vulnerability-fixing commit and these methods can help downstream developers to fix vulnerabilities as early as possible. MegaVul provides a large amount of commit metadata information and code change that can help train deep learning models to automatically identify vulnerability-fixing patches.

5 LIMITATIONS

During the collecting process, we manually selected data sources from websites with more than 50 CVE references, but some websites with less frequent references might still contain potential commits. Besides, some projects use a different version control tool in early development, such as chromium using SVN a long time ago. We do our best to find the corresponding SVN commits in repositories that have been migrated to Git, but some commits may still be missing. During the de-duplication process, some functions with the same functionality may have differences in different branches of the codebase. For example, a function may have compiler-specific macros in a new version of the codebase while the old version lacks it. Although theoretically representing the same function, it is challenging to automatically determine whether their content is identical, resulting in duplication in the dataset. In the future, we will adopt advanced techniques (i.e., NLP and refactoring detection techniques) to further improve data quality.

6 RELATED WORK

Several vulnerability databases for C/C++ have been previously proposed [1, 7, 10, 16, 20]. Alves et al. [7] collected vulnerability code from five widely used libraries. They identified unique vulnerability identifiers (i.e., CVE IDs) and their corresponding commits from the security reporting websites of each library. SARD [1] artificially synthesizes vulnerable code by using well-known vulnerable patterns, which seems to be too simple to reflect the complexity of vulnerability patterns in the real world and is further extended by VulDeePecker [16] with a focus on two specific CWE types (i.e., CWE-119 and CWE-399). Both Devign [20] and Reveal [9] are also collected from several real-world open-source projects. In addition, VulData7 [13], Big-Vul [10], and CVEfixes [8] collect vulnerabilities from NVD databases and find possible vulnerability fixing commits by referencing the links provided in reported CVEs.

7 CONCLUSION

We release a new dataset called **MegaVul**, which automatically extracts reference links from the NVD database for each CVE, mines potential vulnerability fixing commits, and downloads the commit metadata and files from the five popular Git-based web services. MegaVul utilizes an enhanced syntax rule-based *Tree-sitter* to extract high-quality functions and apply several filters to enhance the dataset's quality. Moreover, we have appended additional information to the functions, including abstract functions at nine different granularities, parsed function signatures, and various graph data, which significantly avoid duplication work on code pre-processing for downstream researchers. Furthermore, we provide an interface to support more filters and Git-based platforms, and the implementation of MegaVul can be easily migrated to other programming languages (e.g., Java) with minor code modifications.

In conclusion, MegaVul has gathered high-quality functions from 9,019 commits, including 17,380 vulnerable and 322,168 non-vulnerable functions. All scripts and the dataset are publicly available on GitHub. We will continue to update and expand the dataset to improve its quality and advance research in the field of security.

ACKNOWLEDGEMENTS

This research is supported by the National Natural Science Foundation of China (No. 62202419), the Natural Science Foundation of Zhejiang Province (No. LY24F020008), and the Ningbo Natural Science Foundation (No. 2022J184).

REFERENCES

[1] 2018. Software assurance reference dataset (SARD). https://samate.nist.gov/SARD/.

[2] 2023. Common Vulnerabilities and Exposures. https://cve.mitre.org/.

[3] 2023. Git. https://git-scm.com/

[4] 2023. Joern. https://github.com/joernio/joern

[5] 2023. MegaVul Github Source Code. https://github.com/Icyrockton/MegaVul

[6] 2023. National Vulnerability Database (NVD). https://nvd.nist.gov/.

[7] Henrique Alves, Baldoino Fonseca, and Nuno Antunes. 2016. Software metrics and security vulnerabilities: dataset and exploratory study. In *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 37–44.

[8] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. 30–39.

[9] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2021. Deep learning based vulnerability detection: Are we there yet. *IEEE Transactions on Software Engineering* (2021).

[10] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. A C/C++ code vulnerability dataset with code changes and CVE summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 508–512.

[11] Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. 1987. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 9, 3 (1987), 319–349.

[12] Michael Fu and Chakkrit Tantithamthavorn. 2022. LineVul: A Transformer-based Line-Level Vulnerability Prediction. (2022).

[13] Matthieu Jimenez, Yves Le Traon, and Mike Papadakis. 2018. [Engineering Paper] Enabling the Continuous Analysis of Security Vulnerabilities with VulData7. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 56–61.

[14] Yi Li, Shaohua Wang, and Tien N Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 292–303.

[15] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2021. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing* (2021).

[16] 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. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium*.

[17] Chao Ni, Xin Yin, Kaiwen Yang, Dehai Zhao, Zhengchang Xing, and Xin Xia. 2023. Distinguishing Look-Alike Innocent and Vulnerable Code by Subtle Semantic Representation Learning and Explanation. In *Proceedings of the 2023 31th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM.

[18] Wenbo Wang, Tien N Nguyen, Shaohua Wang, Yi Li, Jiyuan Zhang, and Aashish Yadavally. 2023. DeepVD: Toward Class-Separation Features for Neural Network Vulnerability Detection. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2249–2261.

[19] Xin-Cheng Wen, Yupan Chen, Cuiyun Gao, Hongyu Zhang, Jie M. Zhang, and Qing Liao. 2023. Vulnerability Detection with Graph Simplification and Enhanced Graph Representation Learning. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2275–2286. https://doi.org/10.1109/ICSE48619.2023.00191

[20] 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 *In Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 10197–10207.