2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W) | 978-1-6654-0262-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/DSN-W54100.2022.00032

Vulnerability Dataset Construction Methods Applied To Vulnerability Detection: A Survey

Yuhao Lin[1,3], Ying Li[1, 3], Mianxue Gu[1,3], Hongyu Sun[2,3], Qiuling Yue[1], Jinglu Hu[4], Chunjie Cao[1], Yuqing Zhang[1,2,3]
[1]School of Cyberspace Security, Hainan University, Haikou, China, zhangyq@nipc.org.cn
[2]College of Cyber Engineering, Xidian University, Xi'an, China
[3]National Computer Network Intrusion Protection Center, University of Academy of Sciences, Beijing, China
[4]Graduate School of Information, Production and Systems, Waseda University, Japan

***Abstract*—The increasing number of security vulnerabilities has become an important problem that needs to be solved urgently in the field of software security, which means that the current vulnerability mining technology still has great potential for development. However, most of the existing AI-based vulnerability detection methods focus on designing different AI models to improve the accuracy of vulnerability detection, ignoring the fundamental problems of data-driven AI-based algorithms: first, there is a lack of sufficient high-quality vulnerability data; second, there is no unified standardized construction method to meet the standardized evaluation of different vulnerability detection models. This all greatly limits security personnel's in-depth research on vulnerabilities. In this survey, we review the current literature on building high-quality vulnerability datasets, aiming to investigate how state-of-the-art research has leveraged data mining and data processing techniques to generate vulnerability datasets to facilitate vulnerability discovery. We also identify the challenges of this new field and share our views on potential research directions.**

***Index Terms*—security vulnerabilities, deep learning, datasets**

I. Introduction

A software security vulnerability (abbreviated as a vulnerability) refers to a defect in the operating logic or system security policy of the software [1]. Attackers can exploit vulnerabilities to compromise information systems or access unauthorized information (such as accessing sensitive information, corrupting or modifying data, denial of service, etc.), which leads to a wide range of security and privacy concerns. Furthermore, once a security breach is disclosed, it will have a severe negative impact on the market value of software vendors [2]. Therefore, the direct way to avoid these hazards is to find and fix these security holes as soon as possible before they are exploited by attackers.

The rapid development of machine learning (ML) technology provides a new research direction for automated vulnerability mining [3], [4]. ML algorithms can automatically capture the intrinsic structure of complex data and learn and generalize many abstract features of vulnerable code [5], even latent features that human experts may never consider [1]. Additionally, some information obtained from version control systems (e.g., developer activity [6] and code commits [7]) can also be used by ML for vulnerability detection. Therefore, researchers are keen to use neural models to learn and identify common features of potential vulnerabilities in programs. However, automated detection of security vulnerabilities based on AI models usually relies on high-quality security vulnerability datasets. In the era of large models, the same AI model trained on datasets of different quality may lead to completely different prediction results. In other words, a vulnerability dataset with complete information, reasonable annotation, and a sufficiently large amount of data is a direct factor that affects the performance of AI vulnerability mining. Therefore, the establishment and maintenance of security vulnerability datasets is an important research content in AI vulnerability mining.

In this paper, we first describe some of the problems that may be encountered in the process of constructing a vulnerability dataset. We then review some recent studies that have constructed different vulnerability datasets based on different objectives, aiming to improve the performance of AI vulnerability detection models. Finally, we point out the shortcomings and unresolved issues of existing work, aiming to suggest possible future research directions and potential solutions. In summary, we make the following contributions:

- We analyze the many difficulties of vulnerability dataset construction and summarize some solutions.
- To the best of our knowledge, there are no studies that summarize the construction and quality of vulnerability datasets. This paper summarizes all the recent research in this area in the last decade and analyzes the strengths and weaknesses of some popular vulnerability datasets.
- To facilitate future research, we discuss some of the issues that need to be addressed in existing datasets and describe some of our potential views on these issues.

II. Problems and Some Solutions

Over the past decade, many researchers have proposed various vulnerability datasets in different works, which are often built on real-world programs and based on specific goals. Therefore, other security researchers collect these datasets to train or evaluate their techniques. These datasets are almost always obtained from the world's authoritative vulnerability

Yuqing Zhang is the corresponding author.

databases or patch information provided by software vendors, including the U.S. National Vulnerability Database NVD[1], the Danish vulnerability database Secunia[2], SecurityFocus[3], the Chinese National Information Security Vulnerability Sharing Platform CNVD[4], the Chinese National Information Security Vulnerability Database CNNVD[5], and NSFocus[6]. Of course, there are also a few datasets that are constructed based on manual injection of vulnerabilities or code rewrites. Although these datasets preserve properties such as basic information, characteristics, and solutions of security vulnerabilities in various ways, there are still many challenges in constructing datasets.

A. *Data Source Reliability Issues*

Due to different vulnerability description specifications and information asymmetry, the same CVE-ID vulnerability may have data discrepancies, which can lead to data redundancy or errors in multi-source vulnerability datasets. Dong et al. systematically evaluate the problem of data discrepancy among various vulnerability databases. [8]. The relevant statistical results are shown in Fig. 1, the highest matching rate between each database does not exceed 40. They found that the same vulnerabilities across databases are very different, including different names of the same software project, inconsistent software version ranges affected by vulnerabilities, different vulnerability risk levels, and so on. Table I compares the software versions affected by the CVE-2018-20242 vulnerability in NVD and SecurityFocus. However, for the same CVE-ID vulnerability, which vulnerability library description information is accurate and reliable, it still needs to be checked manually, which undoubtedly brings huge labor consumption. Therefore, it is very important to develop uniform description and retrieval specifications for different data sources.

B. *Multi-Source Data Missing*

When building datasets from multiple data sources, it is expected that the generated datasets will inevitably introduce null values due to missing parts of vulnerable data in the data sources, or even data dimensional differences between different data sources. For example, some CVE records in the NVD database may only contain descriptive information, missing vulnerability types, project repository links, severity scores, etc. Such vulnerability data is considered flawed for studying vulnerabilities using machine learning methods. Rostami et al. propose a supervised machine learning approach to fill in the missing cyberattack behavior classification value (ATT&CK) in most vulnerability samples [9]. The method deduces the corresponding ATT&CK value according to the common attack pattern enumeration and classification (CAPEC) features in the vulnerability data set. This solves the incompleteness problem

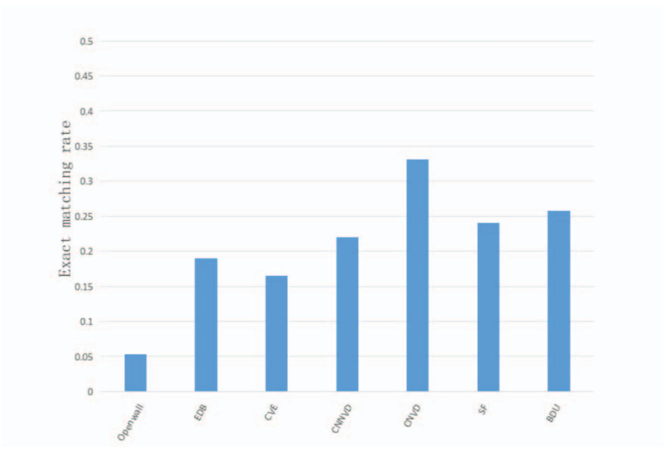

Fig. 1. Exact Match Rate of NVD with Other Vulnerability Databases

of the vulnerability dataset to a certain extent. However, this method still has limitations as it cannot learn categorical features that are not included in the dataset and modeling them is difficult due to the complexity and partial generality of aggressive behaviors.

C. *Feature*

Features are the key to vulnerability model learning (such as risk level, affected version, system environment, etc.). The more features a vulnerability sample contains, the more comprehensive the description of the vulnerability will be, and it will be more conducive to our in-depth research on the vulnerability. For example, the patch signature of a vulnerability specifically describes how the vulnerability is fixed. When detecting code reuse vulnerabilities, AI models can accurately identify the difference between vulnerable code and fixed code, reducing false positives. Xiao et al. proposed a dataset construction method, which combines vulnerability codes with patch codes and stores them in the form of hash values, which can quickly and accurately identify code reuse vulnerabilities [10]. However, most of the existing vulnerability datasets only collect the source code and normal code of the vulnerability, lacking a comprehensive description of the vulnerability [11]. However, considering too many features can also significantly degrade the performance of the model. Therefore, what characteristics should the vulnerability dataset have is a question worth thinking about.

D. *Granularity*

Furthermore, granularity is another important issue to consider during dataset construction. It determines the amount of effective information contained in each data sample. A high-quality dataset should contain multiple granularity levels [12], [13], such as files, classes, functions, etc. Since vulnerability-related code is a small percentage of the overall project, learning too much normal code for a model will cause a trained model to be almost completely unable to identify vulnerabilities in real-world programs. Function-level granularity

[1]NVD, https://nvd.nist.gov/
[2]Secunia, http://secunia.com/
[3]SecurityFocus, http://www.securityfocus.com/vulnerabilities
[4]CNVD, http://www.cnvd.org.cn/
[5]CNNVD, http://www.cnnvd.org.cn/
[6]NSFocus, http://www.nsfocus.net/index.php

TABLE I
EXAMPLE OF CVE-2018-20242 WITH DIFFERENT VERSIONS IN DIFFERENT DATABASES

Vul.DB	Software Version
NVD	JSPWiki:[1.4.0, 1.5.0, 1.5.5 beta, 1.5.7 beta, 1.6.0, 1.6.11 beta, 1.6.12 beta, 1.7.0, 1.8.0, 1.8.2]
SecurityFocus	JSPWiki:[2.10.3, 2.10.4, 2.4, 2.5.139-beta, 2.1.120, 2.1.121, 2.1.122, 2.1.123, 2.4.103, 2.4.104, 2.5.139 beta, 2.5.139]

effectively alleviates the former problem, since most datasets are currently trained with functions as the basic unit of the model [11], [14]. However, the granularity at the function level is still not enough to contain the root cause of the vulnerability, so that the model can only find the vulnerability function, but cannot precisely locate the code directly related to the vulnerability, and cannot deal with cross-function problems. Therefore, the granularity of the vulnerability dataset should be dominated by code fragments, i.e. it must be based on the composition of vulnerable statement slices [15], not just vulnerable functions. Of course, this requires the support of programming language processing techniques. For example, Li et al. used the inter-process program slicing technique to extract vulnerability-related sentences and construct a dataset, which alleviated the cross-function problem of vulnerability mining [16]. [10], on the other hand, used the authors' improved program slicing technique to process the source code to generate statement granularity datasets.

E. Size of Dataset

Some datasets are derived from identified vulnerabilities of NVD or CNVD. Although they provide more complete multi-dimensional features as well as high-quality labels, the number of such samples is very limited and may not be sufficient for model training [17]. Of course, a large vulnerability sample size is not a good thing. More vulnerability data does not mean that the dataset contains more classes of vulnerabilities. Due to the diversity of vulnerabilities in real-world programs, it is almost impossible to train a vulnerability detection model that truly satisfies real-world programs on datasets that only cover one or a few vulnerability types. In addition, we have to consider the catastrophic forgetting problem of vulnerability detection models in the face of ever-expanding datasets.

In response to the lack of real-world vulnerability data sources, researchers have been working hard to develop artificial vulnerability synthetic datasets. Security experts build such synthetic datasets by injecting vulnerabilities into existing programs or by crafting artificially vulnerable programs. Gavitt et al. generated the dataset LAVA-M by modifying the source code of four real-world programs (base64, md5sum, uniq, who) to automatically insert multiple vulnerabilities at once [18]. This dataset has been widely used for performance evaluation of vulnerability detection tools [19]. Wang et al. propose an improved oversampling technique that modifies key statements at the source code level to generate a set of synthetic patch data to form a hybrid dataset with the NVD-based dataset [20]. Their experimental results claim that, for datasets of limited size, synthetic data can improve the accuracy of vulnerability detection models in identifying vulnerabilities to a certain extent.

In general, constructing a unified and standardized comprehensive vulnerability dataset is beneficial to measure the performance differences among different vulnerability detection models, reduce errors, and facilitate in-depth research on security vulnerabilities. We will present the vulnerability datasets proposed by existing studies in the next section.

III. EXISTING STATE-OF-THE-ART VULNERABILITY DATASETS

Many existing studies have created specific datasets based on different criteria and objectives. However, only a few have fully disclosed their datasets. TableII summarizes some of these popular vulnerability datasets, and we systematically compare them in six aspects, as thess are fundamental properties that most vulnerability datasets are built to take into account. Through comparison, we found that most of the current mainstream vulnerability datasets come from open source vulnerability libraries or software project libraries. Although there are also some synthetic vulnerability datasets or even hybrid datasets, the development in this area is still preliminary. Second, a large part of the dataset lacks the necessary patch features. And without considering binary source files, the way most vulnerabilities are labeled relies on source code difference analysis. Below we discuss their pros and cons in detail.

A. Real-World Program Vulnerability Dataset

Zheng et al. proposed a dataset generation method that combines commit history discrepancy analysis and static analysis tools [21]. Unlike existing function-level datasets, the data samples of D2A are derived from inter-procedural analysis and retain more details such as bug types, locations, traces, and parser outputs. In addition, D2A retains the compiler parameters used to compile the relevant source files, which is very useful for dynamic analysis of vulnerabilities.

Considering the high overhead of analyzing successive commit versions, the authors trained the vulnerability identification model CMA based on the NVD vulnerability database to find commit versions and their associated versions that may contain vulnerabilities. However, due to the lack of patch information, the trained CMA model may identify the patched versions as

[7]https://github.com/ibm/D2A
[8]https://github.com/secureIT-project/CVEfjxes
[9]http://samate.nist.gov/SRD/testsuite.php
[10]https://github.com/SunLab-GMU/PatchDB
[11]https://github.com/SAP/vulnerability-assessment-kb/tree/master/MSR2019
[12]https://doi.org/10.5281/zenodo.4734050

TABLE II
OPEN SOURCE DATASETS FOR AI VULNERABILITY MINING

Dataset	Source	Granularity	Released	Vulnerability Type	Patch	Marking Method
D2A[7]	real world	cross-function	√	√	×	multiple static analysis
CVEfixes[8]	real world	mixed	√	√	√	NVD + commit code diff
Juliet[9]	synthetic	function	√	√	×	predefined pattern
PatchDB[10]	mixed	slice	√	√	√	NVD + commit code diff
Ponta et al.[11]	real world	sentence	√	×	√	manually
CPG-based dataset	real world	slice	×	√	×	NVD + SARD
CrossVul[12]	real world	file	√	√	√	NVD + commit code diff

Note: Source: source of the dataset, Granularity: granularity size of the dataset, Released: open source, Vulnerability Type: vulnerability is classified, Patch: include patch information, Marking Method: how each piece of data is properly marked.

vulnerable. Then it is likely to be mislabeled when performing adjacent version difference analysis, resulting in an inaccurate generated dataset [22]. Also, changes in the position of certain functions or invalid commits can cause discrepancy analysis errors. Therefore the quality of this dataset is limited by the reliability of the model.

The multi-class dataset CVEfixes was proposed by Bhandari et al [23]. They forgo simple classification of source code as vulnerable or non-vulnerable, which is common in many vulnerability datasets [24]. CVEfixes classifies vulnerabilities according to categories defined by the Common Weakness Enumeration (CWE) type, while introducing CVSS severity scores as a measure of the vulnerability. This enables a more comprehensive study of the multi-class vulnerability prediction problem. However, its data source is relatively single, and it only obtains vulnerability data from NVD. When some project repositories referenced by NVD are no longer available, the vulnerability code and corresponding repair submissions will not be available from these repositories. Either remove them centrally, or need to do some extra checks to consider other ways to get these fix commits. These all bring challenges to the quality of the dataset.

Ponta et al. manually collected code statement-level data on open-source software vulnerabilities and their fixes, covering 205 different projects, including vulnerabilities not yet indexed by NVD, and vulnerability data obtained from software vendors [25]. Specifically, the dataset includes 29 vulnerabilities without CVE identifiers, and 46 vulnerabilities for which CVE identifiers have been assigned but not yet published in NVD. Compared to automatically constructed datasets (D2A, CrossVul, etc.), it is completely manually generated, and the accuracy of vulnerability labels is high. And the granularity is much finer, down to the statement level, as it maps bugs to bugfix commits. However, the entire dataset construction process lacks automation, and the maintenance and update workload is heavy. And its data volume is very small, which can hardly meet the training needs of vulnerability detection models.

Guan et al. proposed a method to build vulnerability datasets based on code property graph (CPG) to generate high-quality CPG-based datasets [26]. Compared with the source code-based vulnerability dataset, the CPG-based dataset only retains the code attribute graph of the source code, which more completely retains the semantic information of the key code while reducing the data dimension, and significantly reduces noise. At the same time, the dataset has a graph structure, which helps to promote the application of graph neural network in the field of vulnerability mining. However, the quality of this dataset relies on static program slicing technology, which cannot extract semantic information across files, and can only generate code slices for four vulnerability types, lacking generality.

Nikitopoulos et al. provide a dataset of vulnerable code, as well as patch commits retrieved from the open-source software repository Git commits [27]. Similar to D2A, they train a natural language processing model to discover commit messages related to vulnerability patches, which effectively expands the features of the vulnerability dataset. But one of its flaws is that it is split and tagged only for vulnerable files, not in functions. The granularity of the dataset is too coarse, and the entire file containing the vulnerability is directly marked as a vulnerability, which is difficult to achieve a good model training effect.

B. Artificially Synthesized Vulnerability Dataset

The Software Assurance Metrics and Tool Evaluation (SAMATE) project created the Juliet vulnerability dataset as an evaluation metric for static analysis tools designed to identify vulnerabilities in source code [14]. However, Juliet is a synthetic dataset generated from predefined patterns. Although their data size is substantial, they lack diversity compared to real-world programs [12]. If the differences and connections between synthetic datasets and real-world vulnerabilities are not fully understood, blindly using synthetic datasets to evaluate vulnerability detection models, their measurements are unreliable and also lack fairness and impartiality. For example, Wang et al. [28] found that for the synthetic dataset LAVA-M, Angora (an improved fuzz testing tool based on AFL) could detect almost all of its vulnerabilities, and AFL could only detect a very small fraction of them. But for some of the same real-world vulnerabilities, AFL found 62% more vulnerabilities than Angora (e.g., CVE-2017-6966, CVE-2018-11416, CVE 2017-13741).

C. Hybrid Vulnerability Dataset

PatchDB is a large-scale hybrid dataset containing patch information [20]. PatchDB contains not only the validated security patches indexed by NVD, but also a large amount of patch information obtained from the wild. In addition, for vulnerability data diversity, PatchDB provides an additional synthetic dataset that is automatically generated from existing samples by an improved oversampling technique. Their experimental results show that synthetic patches can improve the model's ability to automatically identify security patches for datasets of limited size. But this is operating in vector space, and in the realm of source code, synthetic data is still not interpretable in terms of syntax and semantics.

In addition, we also counted the distribution of existing vulnerability datasets in relation to common CWE vulnerability types and programming languages, as shown in Fig. 2. We found that most vulnerability datasets mainly include vulnerability data of C/C++, followed by Java and PHP, but the sum of the latter is also far less than half of the former. This indicates that the existing vulnerability datasets can barely meet the training requirements of a generic vulnerability detection model across languages. We speculate that this may be related to the widespread use of programming languages. More importantly, even the vulnerability data of C/C++, which has the largest amount of data, does not fully cover the five most common vulnerability types, which puts the applicability of the vulnerability mining model to evaluation to the test.

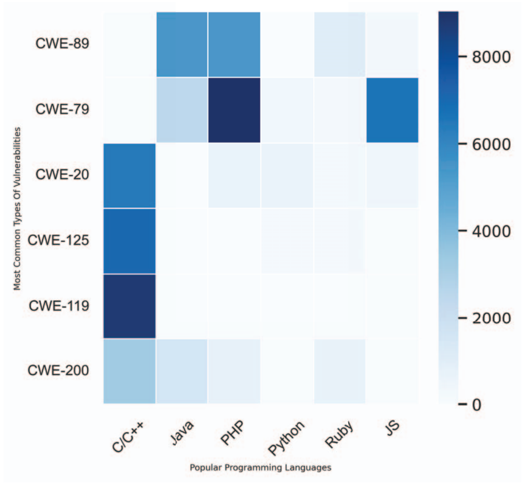

Fig. 2. Distribution of the Five Most Common Types of Vulnerabilities and Popular Programming Language.

In summary, while existing open source vulnerability datasets can meet the training needs of specific AI models to some extent, they still have obvious limitations and cannot provide a comprehensive overview of the real-world multivariate vulnerability landscape. To the best of our knowledge, there is no dataset that meets the criteria of a true benchmark dataset.

IV. Challenges and Opportunities

A. Benchmark Dataset

By reviewing the existing research on vulnerability datasets, we find that most vulnerability datasets are specially generated for specific vulnerability detection models [17], [11], lack scalability, and cannot horizontally evaluate different performance between models. Not only is the data source single, but most of them only contain C/C++ open source projects, which are not comprehensive enough [16], [29], [14]. The value of any vulnerability dataset decreases over time as the performance of vulnerability detection models improves. Also, some vulnerability detection models perform well on synthetic datasets, but they may perform poorly in detecting real vulnerabilities [14], [28]. However, the number of vulnerabilities in real-world programs is huge and very diverse, so constructing a reasonable dataset from numerous vulnerability data is a major challenge. A dataset with diverse data sources, a sufficiently large amount of data, comprehensive vulnerability characteristics, evenly distributed vulnerability types and frequent updates is crucial for measuring the performance of vulnerability detection models. Therefore, a standardized and normalized benchmark dataset is urgently needed as a unified standard for evaluating and comparing model effectiveness. We believe that standard benchmark datasets should be carefully tailored based on multi-level detection granularity, and should contain both a small number of synthetic and a large number of real-world code samples for qualitative and quantitative evaluation and comparison.

B. Automated Dataset Labeling Framework

Furthermore, most of the available vulnerability datasets are usually small in scale, and a large portion of these datasets lack labeling information or are even mislabeled [30], [31], [5]. If not properly labeled, this data can become a noise point that affects the performance of the detection model. Currently, most automated vulnerability data collection frameworks either rely on the prior knowledge of security experts to determine the correct labeling of samples, or assume that all collected samples are correctly labeled, which is clearly unreasonable. Manual annotation seems to solve part of the problem, but manual verification of each sample data is a very inefficient method. Therefore, AI model-based automated vulnerability data labeling combined with human review may be more practical for improving the quality of vulnerability datasets. We believe this is an interesting direction for future vulnerability dataset research.

C. Security and Non-security Commits

Another important issue is that commits that are referenced as vulnerability fixes may still have other vulnerabilities that have not yet been fixed, and even patches may not work. This means that there may be similar commits in the future, or the commits may contain code changes that are not related to the vulnerability fix, such as documentation improvements and new function additions. To further improve the quality of the dataset, variance analysis of consecutive commits is essential. But how to accurately find commits regarding vulnerability fixes and patches from the large number of project commits will help reduce the analysis ground overhead. It would be an interesting task to accurately distinguish between patches and non-vulnerable modified code information, or even failed patches.

V. Conclusion

The application of artificial intelligence in vulnerability detection is a promising research direction, but there are still many unresolved issues. In this paper, we discuss the problems faced in vulnerability dataset construction, review relevant research on datasets applied to AI vulnerability detection, show the trends of real-world program-based vulnerability datasets and synthetic vulnerability datasets, and analyze their advantages and disadvantages, aiming to promote scholars in the security field to reach a unified standard and specification for vulnerability dataset construction as soon as possible. This will motivate and attract more researchers to contribute to this promising field.

Acknowledgment

This work was supported by the Key Research and Development Science and Technology of Hainan Province (ZDYF202012, GHYF2022010), and the National Natural Science Foundation of China (U1836210).

References

[1] C. D. Sestili, W. S. Snavely, and N. M. VanHoudnos, "Towards security defect prediction with ai," *arXiv preprint arXiv:1808.09897*, 2018.

[2] R. Telang and S. Wattal, "Impact of software vulnerability announcements on the market value of software vendors-an empirical investigation," *Available at SSRN 677427*, 2005.

[3] S. Cao, X. Sun, L. Bo, Y. Wei, and B. Li, "Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection," *Information and Software Technology*, vol. 136, p. 106576, 2021.

[4] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: A survey," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1825–1848, 2020.

[5] G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, O. De Vel, and P. Montague, "Cross-project transfer representation learning for vulnerable function discovery," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3289–3297, 2018.

[6] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of linus' law," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 453–462.

[7] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 426–437.

[8] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang, "Towards the detection of inconsistencies in public security vulnerability reports," in *28th USENIX Security Symposium*, 2019, pp. 869–885.

[9] S. Rostami, A. Kleszcz, D. Dimanov, and V. Katos, "A machine learning approach to dataset imputation for software vulnerabilities," in *International Conference on Multimedia Communications, Services and Security*, 2020, pp. 25–36.

[10] Y. Xiao, B. Chen, C. Yu, Z. Xu, Z. Yuan, F. Li, B. Liu, Y. Liu, W. Huo, W. Zou *et al.*, "Mvp: Detecting vulnerabilities using patch-enhanced vulnerability signatures," in *29th USENIX Security Symposium*, 2020, pp. 1165–1182.

[11] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *arXiv preprint arXiv:1909.03496*, 2019.

[12] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, 2015, pp. 1–9.

[13] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "μvuldeepecker: A deep learning-based system for multiclass vulnerability detection," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[14] T. Boland and P. E. Black, "Juliet 1.1 c/c++ and java test suite," *IEEE Computer Society*, vol. 45, no. 10, pp. 88–90, 2012.

[15] M.-j. Choi, S. Jeong, H. Oh, and J. Choo, "End-to-end prediction of buffer overruns from raw source code via neural memory networks," *arXiv preprint arXiv:1703.02458*, 2017.

[16] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[17] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681*, 2018.

[18] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. Robertson, F. Ulrich, and R. Whelan, "Lava: Large-scale automated vulnerability addition," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 110–121.

[19] Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, "Steelix: program-state based binary fuzzing," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 627–637.

[20] X. Wang, S. Wang, P. Feng, K. Sun, and S. Jajodia, "Patchdb: A large-scale security patch dataset," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 149–160.

[21] Y. Zheng, S. Pujar, B. Lewis, L. Buratti, E. Epstein, B. Yang, J. Laredo, A. Morari, and Z. Su, "D2a: A dataset built for ai-based vulnerability detection methods using differential analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 111–120.

[22] S. Suneja, Y. Zheng, Y. Zhuang, J. A. Laredo, and A. Morari, "Towards reliable ai for source code understanding," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, p. 403–411.

[23] G. Bhandari, A. Naseer, and L. Moonen, "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*, 2021, p. 30–39.

[24] ——, "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*, 2021, pp. 30–39.

[25] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi, and C. Dangremont, "A manually-curated dataset of fixes to vulnerabilities of open-source software," in *Proceedings of the 16th International Conference on Mining Software Repositories*, 2019, p. 383–387.

[26] Z. Guan, X. Wang, W. Xin, and J. Wang, "Code property graph-based vulnerability dataset generation for source code detection," in *Frontiers in Cyber Security*, 2020, pp. 584–591.

[27] G. Nikitopoulos, K. Dritsa, P. Louridas, and D. Mitropoulos, "Crossvul: A cross-language vulnerability dataset with commit data," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, p. 1565–1569.

[28] H. Wang, X. Xie, Y. Li, C. Wen, Y. Li, Y. Liu, S. Qin, H. Chen, and Y. Sui, "Typestate-guided fuzzer for discovering use-after-free vulnerabilities," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 999–1010.

[29] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "Vulinoss: a dataset of security vulnerabilities in open-source systems," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 18–21.

[30] G. Lin, J. Zhang, W. Luo, L. Pan, O. De Vel, P. Montague, and Y. Xiang, "Software vulnerability discovery via learning multi-domain knowledge bases," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[31] G. Lin, J. Zhang, W. Luo, L. Pan, and Y. Xiang, "Poster: Vulnerability discovery with function representation learning from unlabeled projects," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2539–2541.