Automated Software Vulnerability Collection for a Database with Static Information

João Rafael Henriques and José D'Abruzzo Pereira

University of Coimbra, Centre for Informatics and Systems of the University of Coimbra, Department of Informatics Engineering
joaohenriques@student.dei.uc.pt,josep@dei.uc.pt

Keywords: software vulnerability, security, static code analysis, software metrics

Software vulnerabilities are present in most software applications that are used on a daily basis. It is widely known that software vulnerabilities are a problem for software applications as they are an open door to attacks [1]. When an attack occurs, it can cause severe consequences, such as damage to the operation and cause legal and financial implications.

Fighting against this and building secure applications is a very complex job, and, throughout the years, several techniques were created to detect these vulnerabilities, allowing the software development team to make possible their correction [2]. Such techniques usually rely either on static or dynamic techniques.

Static techniques do not require the source code to be run to identify software vulnerabilities. The most used static technique is Static Code Analysis (SCA) using Static Analysis Tools (SATs). On the other hand, dynamic techniques require the source code to be run to identify the vulnerabilities. A widely known dynamic technique is software penetration testing.

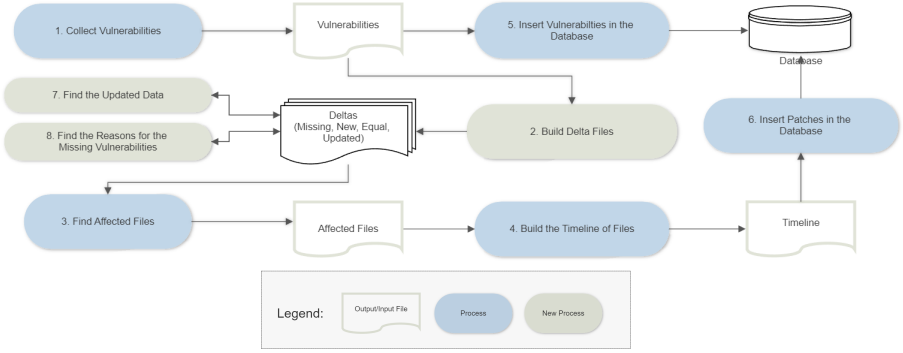
Although these techniques help identify the vulnerabilities, all of them suffer from the same issue: either do not detect all the vulnerabilities (false negatives) or report items that are not actual vulnerabilities (false positives). Consequently, the software development teams spend some time analyzing the reported items without knowing that all vulnerabilities have been reported.

To support the creation and improvement of vulnerability detection techniques, there are vulnerability datasets available [3,4,5]. A methodology to collect software vulnerabilities was proposed in a previous work [6]. However, it suffers the same problem as other software vulnerability databases: their data are usually frozen and are not frequently updated with newly reported vulnerabilities. Hence, we propose a solution that automates the vulnerability collection.

We automated the scripts previously created [6] by using `Crontab`. The automation will update the data about the vulnerabilities of open-source C/C++ projects, Mozilla, Kernel, Glibc, Httpd, and Xen. The process starts collecting all the vulnerabilities from the website *www.cve-details.com* [7], which contains vulnerability information about the selected projects. Then, we find the files changed for each vulnerability fix in the repositories. This step takes a lot of time to conclude. Using the identifier (hash) of each commit, it is possible to

find the changes that occurred and a list of files, as well as the lines that were changed. With the lines changed, we can obtain the functions and classes that were modified. A timeline of every file since the first commit of the repositories is done next. This is an important and complex task because the repositories are not linear, as they follow a tree structure. Finally, the vulnerabilities and patches are inserted into the database, and the information is available to everyone with access to the dataset. A summary of the collection process can be seen in Fig. 1.

Fig. 1. Pipeline to collect the vulnerabilities and static information



An up-to-date dataset can support studies reflecting the most recent vulnerabilities. Doing the collection every day and running every step every day takes a lot of time and resources. To improve the performance and to allow the execution of the vulnerability collection daily, we designed and built an algorithm to separate new and updated information. It is useless to search the affected files of a vulnerability that does not have changes compared to the previous day. Because of that, all vulnerabilities collected daily are classified as *i)* **Updated**, *ii)* **New**, *iii)* **Equal**, or *iv)* **Miss** vulnerabilities. The next steps are only executed in the first two sets of vulnerabilities to improve the collection performance. The scripts can now be run every day.

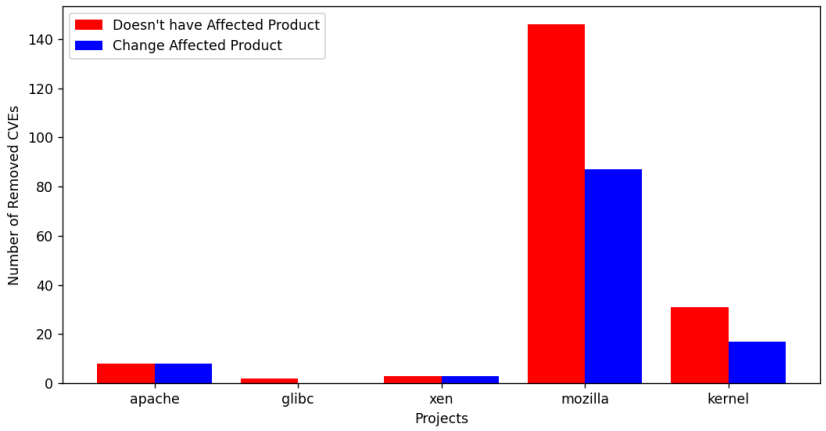
It was not expected that some vulnerabilities would not be reported every day through our collection mechanism. Hence, we needed to investigate the possible reasons for that. Results show that the product field (from CVE Details) is the one with more changes in the vulnerabilities already present in the database. This analysis helps us answer the following Research Question (RQ) *RQ1: What are the main changes in the software vulnerabilities of these projects between 2019 to 2022?*.

This field is used to relate the vulnerabilities to the projects, and if some change occurs in that place, the algorithm can associate the vulnerability with a different project. When this occurs, the vulnerabilities are not detected by our scripts. So, we built a way to find if the vulnerability has new products or was just removed. In the end, we have up-to-date information about all the

vulnerabilities. The changes on the updated vulnerabilities are found and saved to be used later.

These changes can happen for multiple reasons, like a new resolution, or fix for one vulnerability, or removing the product field value that identifies the target project. This is the answer to the *RQ2: What are the main reasons that led to changes in the vulnerabilities?* Everyone can submit a vulnerability on the website, and, with time, every vulnerability will receive updates until it is stable. The main changes can be seen in Fig. 2.

Fig. 2. Changes on the "Affected Products" field



To save the information, we use the database from *Pereira et al.* [6]. All the vulnerabilities detected by the pipeline of scripts are stored in the database with some information about each one. All of them are linked to the respective project (Mozilla, Kernel, Httpd, Xen, Glibc) as well as the location where they are on the repository. Additionally, the commit where the vulnerability is present and the files, functions, and classes about the vulnerability are stored in the database. The vulnerabilities have a CWE (Common Weakness Enumeration) identifier that is used to assign to vulnerability categories, which were proposed by the *Pereira et al.* [8].

The database also contains alerts from two different open-source SATs: CppCheck and Flawfinder. An alert is an item reported by an SAT that may indicate the presence of an issue, such as a vulnerability. Some alerts can be false positives. The database also stores alerts for vulnerabilities from the database.

To conclude, we present an automatic way to use the collection process of vulnerabilities. We optimize the process to maintain the dataset updated as soon as possible. Extracting software metrics from this data and inserting them into the database is also possible and generate alerts and save them as well.

Future work includes the creation of a dashboard to provide information about the vulnerabilities and the main changes they suffer daily. Additionally,

we should also highlight the items that are no longer a vulnerability without removing them from the database. Finally, an updated architecture based on containers should also be developed.

Acknowledgements

This work was partially funded by FCT grant 2020.04503.BD. This work is partially funded by the FCT - Foundation for Science and Technology, I.P. / MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020. This work has been partially supported by the project **AIDA** - Adaptive, Intelligent and Distributed Assurance Platform (reference `POCI-01-0247-FEDER-045907`) leading to this work is co-financed by the ERDF and COMPETE 2020 and by the FCT under CMU Portugal. It is also partially supported by Project **Agenda Mobilizadora Sines Nexus**. ref. No. 7113), supported by the Recovery and Resilience Plan (PRR) and by the European Funds Next Generation EU, following Notice No. 02/C05-i01/2022, Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation.

References

1. M. C. Sánchez, J. M. C. de Gea, J. L. Fernández-Alemán, J. Garceran, and A. Toval, "Software vulnerabilities overview: A descriptive study," *Tsinghua Science and Technology*, vol. 25, no. 2, pp. 270–280, 2020.
2. B. Liu, L. Shi, Z. Cai, and M. Li, "Software Vulnerability Discovery Techniques: A Survey," in *2012 Fourth International Conference on Multimedia Information Networking and Security*, Nov 2012, pp. 152–156.
3. 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.
4. J. Fan, Y. Li, S. Wang, and T. N. Nguyen, *A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries*. New York, NY, USA: Association for Computing Machinery, 2020, p. 508–512. [Online]. Available: https://doi.org/10.1145/3379597.3387501
5. 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," in *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, 2018. [Online]. Available: https://doi.org/10.14722%2Fndss.2018.23158
6. J. D. Pereira, J. H. Antunes, and M. Vieira, "A Software Vulnerability Dataset of Large Open Source C/C++ Projects," in *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2022, pp. 152–163.
7. "Cve details - the ultimate security vulnerability datasource," https://www.cvedetails.com/, 2023, accessed: 2023-05-11.
8. J. D'Abruzzo Pereira and M. Vieira, "On the use of open-source c/c++ static analysis tools in large projects," in *2020 16th European Dependable Computing Conference (EDCC)*, 2020, pp. 97–102.