Benchmarking the vulnerability detection capabilities of software analysis tools

Elena Baninemeh[1], Slinger Jansen[1,2]

[1]*Utrecht University, Utrecht, The Netherlands*
[2]*Lappeenranta University, Lappeenranta, Finland*

Abstract

Code cloning and copy-pasting code fragments is common practice in software engineering. If security vulnerabilities exist in a cloned code segment, those vulnerabilities may spread in the related software, potentially leading to security incidents. Code similarity is one effective approach to detect vulnerabilities hidden in software projects. However, due to the complexity, size, and diversity of source code, current methods suffer from low accuracy, and poor performance. Moreover, most existing clone detection techniques focus on a limited set of programming languages in the detection process. We propose to solve these problems using SearchSECO, a software analysis tool that detects vulnerabilities in multiple programming languages.

Keywords

Software vulnerability, code clone detection, software security, open-source software

1. Introduction

The rapidly growing demands for software lead to the increasing popularity of code reuse, including existing code templates and components. Open-source software (OSS) has become one of the best solutions to improve both the efficiency and the quality of development at the meanwhile reducing cost. However, a considerable number of vulnerabilities in OSS programs would naturally lead to many software vulnerabilities caused by code cloning, which poses a severe threat to system security [1]. In fact, OSS has increased the rate of vulnerabilities because, as the name implies, the code is open-source and available to everyone. Most software developers copy the code from other software systems and reuse them without significant modification. This type of reuse code is called code cloning [2]. Code cloning is expected to rise, especially with tools such as GitHub co-pilot, which uses code templates and auto-completion features to support software engineers.

Information about known vulnerabilities is published through different resources such as the National Vulnerability Database (NVD) in the form of Common Vulnerabilities and Exposures (CVE). Existing techniques for vulnerable code clone detection fall into two categories: code similarity and functional similarity. In code similarity approaches, the target source code is

compared against a set of known vulnerable code samples and determined to be vulnerable if a threshold of similarity is met. Code similarity approaches are typically classified based on four types of detection coverage. type-1 (identical), type-2 (syntactically equivalent), type-3 (syntactically similar), and type-4 (semantically similar) [3].

On the other hand, functional similarity approaches seek to generate abstract functional patterns of code which model vulnerable behavior. However, due to the complexity of building such a pattern, these techniques are typically specialized to only a small class of vulnerabilities or a particular source code project, rendering them ineffective as general-purpose vulnerable code clone detection techniques [4].

In this work, we introduce SearchSECO, a code-similarity technique capable of identifying modified vulnerable code clones while remaining generic to type-1 and type-2 and supporting multiple languages. Additionally, we built a database by mining vulnerable and patched source code from GitHub. In this paper, we present the main two processes, including vulnerabilities collection and vulnerabilities detection.

2. Research Approach

SearchSECO is a large database of methods of the top rated projects (with "stars") on Github. SearchSECO clones a git project, extracts a number of versions, and extracts the files and authors from those versions. The method's abstract syntax tree is extracted and a representation of this abstract syntax tree is hashed. SearchSECO currently parses Java, Javascript, C/C++, and Python. SearchSECO is itself a project on Github and can be found via: https://github.com/SecureSECO/SearchSECOController. Furthermore, the database can be accessed through a portal: https://secureseco.science.uu.nl/portal/. In this portal visitors can enter their own project link and email address. After the project has been processed and matched, the visitor receives a report of the matches in the SearchSECO database and can determine if there are any potentially vulnerable fragments in their project. Currently (June 14th 2022), the database contains 16 million unique methods from approximately 100 thousand projects from Github.

The database until recently only had matching capability, but as it is the meta-data that makes the method database interesting, we have started by matching vulnerability data from vulnerability database and directly from open source project. In this paper, our research objective is to benchmark SearchSECO's performance to other tools. It must be noted that software engineers using SearchSECO are not time constrained. However, they do care about accurate feedback about their projects and therefore we benchmark SearchSECO against other available state-of-the-art tools in terms of precision, rather than performance speed.

The main research question in this study is as follows:(MRQ) How effective is the vulnerability detection feature of SearchSECO compared to other vulnerability detection approaches? We formulated the following research questions to address the MRQ: SRQ_1: Is detection reporting in vulnerability detection approaches sufficient for benchmarking? SRQ_2: How can SearchSECO be compared accurately to other vulnerability detection approaches? SRQ_3: How is the scalability of SearchSECO in detecting vulnerabilities compared to state-of-the-art approaches? SRQ_4: How effective is SearchSECO in detecting the latest vulnerabilities published by CVE and GitHub?

We employed a literature study using the snowballing method, combined with document

Table 1

An overview of the four research methods used in this study with their corresponding research questions (RQ)

Research Method	MRQ	RQ1	RQ2	RQ3	RQ4
Literature study	☒	☒	☒	☒	☒
Document analysis		☒	☒		
Replication study	☒	☒	☒		☒
Benchmark study	☒		☒	☒	

analysis and replication study, and performing an experiment to compare SearchSECO with other vulnerability detection tools. Table 1 shows the mapping between the research questions and the research methods. The preferred literature study method is *snowballing*. Wohlin [5] presents several guidelines for this method which will use during the literature study. *Document analysis* is a systematic procedure for reviewing or evaluating documents, including manuscripts and illustrations, that have been published without a researcher's intervention [6]. Document analysis is one of the analytical methods in qualitative research that requires data investigation and interpretation to elicit meaning, gain understanding, and develop empirical knowledge [7]. The preferred literature study method is *snowballing*. Wohlin [5] presents several guidelines for this method which will use during the literature study. *Document analysis* is a systematic procedure for reviewing or evaluating documents, including manuscripts and illustrations, that have been published without a researcher's intervention [6]. Document analysis is one of the analytical methods in qualitative research that requires data investigation and interpretation to elicit meaning, gain understanding, and develop empirical knowledge [7]. Like many other empirical disciplines, replication has been seen as an essential means of assessing reliability and confidence in empirical findings. A key component of experimentation is replication. To consolidate a body of knowledge built upon experimental results, they must be extensively verified. This verification is carried out by replicating an experiment to check if its results can be reproducible [8]. We aim to use the ACM SIGSOFT Empirical Standards [1] for benchmarking procedure.
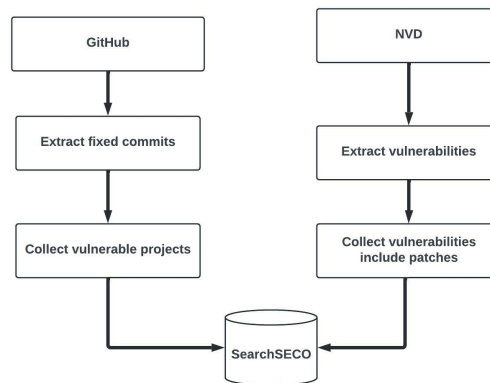
3. SearchSECO

In this section, we describe the design and implementation of our proposed approach Search-SECO for method-level vulnerability detection. We aim to accurately discover the code clones between a set of vulnerable codes and a target program using the code clone detection technique. In SearchSECO, we focus on the detection of vulnerable code fragments accurately, Scaling to a large code base, and Supporting multiple languages.

---

[1]https://github.com/acmsigsoft/EmpiricalStandards/blob/master/docs/Benchmarking.md

3.1. Vulnerabilities collection process

We collected vulnerability data of each project from two sources: NVD and public Git repositories on GitHub. NVD is a vulnerability database built upon and fully synchronized with the CVE list. In addition to a large amount of vulnerability data, it also provides enhanced information (e.g., vulnerability type, references to solutions) for each record. GitHub provides a larger quantity and wider variety of code, which can help us supplement the vulnerability dataset. We built the SearchSECO vulnerability database in the following steps:

- We crawled all of the vulnerability entries in the CVE database and NVD, such as the descriptive information for each vulnerability. Specifically, we parse the Github web pages to extract CVE Details such as vulnerable lines and hash commits.
- The NVD receives its vulnerability listings directly from the CVE. Therefore, vulnerabilities that are not reported to the CVE, so they would not publish in the NVD. Hence, beside extracting vulnerabilities from NVD, we have to extract vulnerabilities from Github (See figure 1). First, SearchSECO clone the repository by using the "git clone repository" command. Then, it will search for the commits regarding CVEs for each repository by using the " git log –grep="CVE-20" command. This process of collecting vulnerable code and extraction required data is fully automated,

Figure 1: This figure shows the vulnerability collection to store in SearchSECO.

3.2. Vulnerabilities detection process

In this section, we describe our approach to vulnerability detection, which is a scalable approach to code clone detection. The types of code clones have to be clarified in order to explain the process. Four different types of code clones are, Type-1: Exact clones, Type-2: Renamed clones, Type-3: Restructured clones, and Type-4: Semantic clones.
Type-1: Identical code fragments, but may have some variations in whitespace, layout, and comments.

Type-2: Syntactically equivalent fragments with some variations in identifiers, literals, types, whitespace, layout, and comments.

Type-3: Syntactically similar code with inserted, deleted, or updated statements.

Type-4: Semantically equivalent but syntactically different code.

We designed SerachSECO to detect Type-1 and Type-2 clones because our goal is to reduce false positives and negatives and increase scalability.

3.2.1. Code clone detection

Figure 2 Shows all the steps and the process of SearchSECO. SearchSECO preprocesses a target program and generates a hash value by the MD5 algorithm. And then, it detects code clones by comparing two or more hashes. By generating a hash value consisting of vulnerable functions and comparing the stored hashes in SearchSECO with a generated hash from the target program, SearchSECO will declare vulnerable code clones in the target program.

3.2.2. Prepossessing

The following steps will perform in the preprocessing when SearchSECO receives the code fragment or project to detect vulnerabilities.

1. Method extraction: The process start with retrieving functions from a given program by using a robust parser.

2. Abstraction and normalization: we used an abstraction and normalization feature, so every formal parameters, local variables, data types, and function calls that appear in the body of a function are replaced with symbols such as FPARAM in level 1, LVAR in level 2, DTYPE in level 3, and FUNCCALL in level 4.

3. Generating hash value: In this step, the hash value generate based on the MD5 algorithm.

4. Related work

Many approaches have been proposed to detect the vulnerabilities brought by code clones. Kim et al. [9] proposed VUDDY, a highly efficient method for detecting vulnerable code cloning, which is achieved by leveraging function-level granularity and a length-filtering technique that reduces the number of signature comparisons. However, it does not support common code modification methods such as word order modification and redundant code insertion, which causes its limitation in practice. VFDETECT [10] proposed an approach based on an innovative fingerprint model to detect vulnerable code. VulPecker [11] developed a technique that identifies a vulnerability-to-similarity-algorithm mapping. This way, each algorithm can be applied to the vulnerabilities to which they are best suited. However, this approach is still limited by the underlying accuracy of the similarity algorithms and only achieves a recall score of 60%, meaning many vulnerable clones were left undetected. VCIPR [12] is a scalable system for vulnerability detection in unpatched source code. That uses a fast, token-based approach to detect vulnerabilities at function level granularity.

Akram and Luo [13] developed a quantitative vulnerability detection technique based on the code clone detection technique at the source code level. They retrieved vulnerable source code
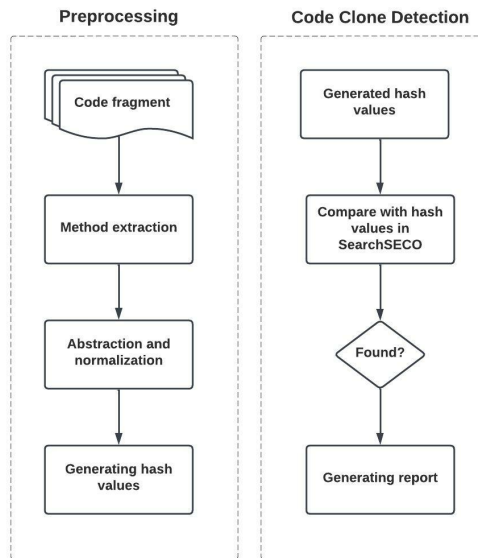
Figure 2: This figure shows the vulnerability collection to store in SearchSECO.

files from the various web source code repositories by tracking the patch file of vulnerabilities. Then, the vulnerable source code files are retrieved using common vulnerabilities and exposures (CVE) numbers.

ReDeBug [14] is a technique that does use the information in both the vulnerable code and the patched code. ReDeBug performs sequence-based matching utilizing the diff files associated with a particular vulnerability. A diff file contains the lines that were explicitly modified during the transition of the code from vulnerable to patched, as well as some context code within close textual proximity. This allows ReDeBug to detect some type-3 clones; however, if the code modification is near the location of the lines modified during the patch process, this technique will fail to detect the vulnerable clone.

5. Conclusion

In this study, we propose a vulnerability detection tool to benchmark with different approaches and methodology from state-of-the-art research on vulnerability detection. We aim to design our approach for scalable and accurate detection of vulnerable code clones. Moreover, we aim to address an automated way to collect vulnerable functions and implement SearchSECO to demonstrate its efficacy and effectiveness to detect numerous vulnerable clones from a large code base with unprecedented scalability and accuracy.

References

[1] J. Guo, H. Li, Z. Wang, L. Zhang, C. Wang, A novel vulnerable code clone detector based on context enhancement and patch validation, Wireless Communications and Mobile Computing 2022 (2022).

[2] M. Mondal, C. K. Roy, K. A. Schneider, Identifying code clones having high possibilities of containing bugs, in: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), IEEE, 2017, pp. 99–109.

[3] C. K. Roy, J. R. Cordy, R. Koschke, Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, Science of computer programming 74 (2009) 470–495.

[4] F. P. Viertel, W. Brunotte, D. Strüber, K. Schneider, Detecting security vulnerabilities using clone detection and community knowledge., in: SEKE, 2019, pp. 245–324.

[5] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th international conference on evaluation and assessment in software engineering, 2014, pp. 1–10.

[6] G. A. Bowen, Document analysis as a qualitative research method, Qualitative research journal (2009).

[7] J. Corbin, A. Strauss, Basics of qualitative research: Techniques and procedures for developing grounded theory, Sage publications, 2014.

[8] N. Juristo, O. S. Gómez, Replication of software engineering experiments, in: Empirical software engineering and verification, Springer, 2010, pp. 60–88.

[9] S. Kim, H. Lee, Software systems at risk: An empirical study of cloned vulnerabilities in practice, Computers & Security 77 (2018) 720–736.

[10] Z. Liu, Q. Wei, Y. Cao, Vfdetect: A vulnerable code clone detection system based on vulnerability fingerprint, in: 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC), IEEE, 2017, pp. 548–553.

[11] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, J. Hu, Vulpecker: an automated vulnerability detection system based on code similarity analysis, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, 2016, pp. 201–213.

[12] J. Akram, L. Qi, P. Luo, Vcipr: vulnerable code is identifiable when a patch is released (hacker's perspective), in: 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), IEEE, 2019, pp. 402–413.

[13] J. Akram, P. Luo, Sqvdt: A scalable quantitative vulnerability detection technique for source code security assessment, Software: Practice and Experience 51 (2021) 294–318.

[14] J. Jang, A. Agrawal, D. Brumley, Redebug: finding unpatched code clones in entire os distributions, in: 2012 IEEE Symposium on Security and Privacy, IEEE, 2012, pp. 48–62.