Securing Software Ecosystems through Repository Mining

Aminul Didar Islam[1,2*], Slinger Jansen[2]

[1]LUT University, Yliopistonkatu 34, 53850 Lappeenranta, Finland
[2]Utrecht University, Utrecht, The Netherlands

Abstract

Through the incessant reuse of code fragments, the worldwide software ecosystem has become highly connected. This provides advantages, such as faster software engineering, however, it also provides new challenges, such as easier spreading of vulnerabilities. The world depends on software and the proliferation of code also causes the proliferation of vulnerabilities. In this PhD project, we explore the use of a code clone hashing and storing technique to enable fast searches of abstract code clones in the worldwide software ecosystem, called SearchSECO. With SearchSECO, we can rapidly identify code, code clones, vulnerabilities, license conflicts, and other aspects of code cloning. With SearchSECO as a platform, we hope to move forward the art and science of repository mining.

Keywords

Code clones, Repository mining, Code identification, License violations, Software engineering

1. Introduction

The worldwide software ecosystem [1] (SECO) concerns all software producing organizations and individuals that collaboratively serve a market for software and services. SECOs comprises a network of developers, vendors, consumers, and other stakeholders who interact through various platforms. It includes open-source repository software and also proprietary software ranging from small to large-scale enterprise software systems [2].

A SECO represents a set of actors that function together as a unit that interacts and communicates with a shared software market according to the services and their relationships [3]. Building on this concept, this research proposes a software provenance theory, ensuring that the origin and history of every software engineering artifact are traceable across the entire software supply network.

Mining Software Repositories (MSR) is a research area within software engineering that focuses on analyzing the vast amount of data generated during software development, maintenance, and usage [4]. This data is stored in various repositories such as version control systems (e.g., GitHub), issue trackers (e.g., Jira), and code review systems (e.g., Gerrit), among others. The main goal of MSR is to extract actionable insights and patterns that can improve software quality, guide development processes, and inform decision-making within software teams.

In the broader context of software engineering, MSR plays a crucial role in supporting evidence-based decision-making. By applying techniques from data mining, machine learning, and information retrieval to software repositories, MSR helps researchers and practitioners understand trends, predict future issues, improve software processes, and evaluate the effectiveness of different practices.

Traditionally, repo mining has focused on file-level or project-level data, providing a broad view of software systems and their evolution [5]. However, as software ecosystems grow in complexity, a finer-grained approach is increasingly necessary. By analyzing source code at the method level, researchers can obtain more detailed insights into code reuse, identify security vulnerabilities with greater precision, and understand the intricacies of software dependencies and maintenance challenges [3].

The 15th International Conference on Software Business (ICSOB 2024), NOVEMBER 18-20, 2024, Utrecht, The Netherlands
*Corresponding author.
aminul.islams@lut.fi (A. Didar Islam); slinger.jansen@uu.nl (S. Jansen)
https://yamadharma.github.io/ (A. Didar Islam); https://kmitd.github.io/ilaria/ (S. Jansen)
0000-0002-0877-7063 (A. Didar Islam); 0000-0003-3752-2868 (S. Jansen)

We propose SearchSECO, a hash based index for code fragments that enables searching source code at the method level in the worldwide software ecosystem [3]. Currently, it is possible to identify files by their hashes in the Software Heritage Graph. We want to create a set of parsers that extract fragments (methods) from the code files and makes them findable. By making methods from the worldwide software ecosystem findable, we can perform more reliable license checks, search for vulnerabilities, and extract call graphs from those methods [6].

We unearth the relationships between code fragments, code files, and their projects on a worldwide scale. This fine-grained data enables much richer analyses, significantly moving forward the field of empirical software engineering and its sub-field of repository mining. Our first projects will deal with license violation detection in open source, vulnerability finding, and software package identification for SBOM creation [6, 7].

2. Research Aim and Impact

The primary aim of this research is to enhance the capabilities of repository mining through the development and deployment of SearchSECO, a hash-based index for code fragments [7]. By enabling method-level source code searching, this research seeks to address critical issues in license conflict detection and vulnerability benchmarking within the worldwide software ecosystem. The goal is to provide a robust, scalable tool that can significantly improve the accuracy and efficiency of software license compliance and security vulnerability detection.

Impact of the Project: The project's main predicted impacts are: 1) enabling data-driven analysis of software ecosystems, and potentially faster submission to the repository mining community, 2) code license checking and conflict identification will be easier and faster with the cutting-edge technology, 3) the outcome of this research will help for better code quality and better findability 4) promotion of the SearchSECO tool to researchers and SearchSECO will ease of use.

2.1. Research Questions and Justifications

RQ1: How effective is SearchSECO in detecting method-level license conflicts compared to existing file-level tools?
Justification: This question aims to evaluate the accuracy and reliability of method-level analysis in identifying license conflicts. It seeks to demonstrate the advantages of finer granularity in code analysis over traditional file-level approaches, which can overlook conflicts that occur within individual methods.
RQ2: What impact does method-level vulnerability benchmarking have on identifying and mitigating security vulnerabilities in software projects?
Justification: This question explores the effectiveness of the vulnerability benchmarking framework developed using SearchSECO. It assesses whether method-level benchmarking provides more detailed and actionable insights into security vulnerabilities, leading to improved detection and mitigation strategies.
RQ3: How can the relationships between code fragments, files, and projects be leveraged to improve software maintenance and evolution?
Justification: This question investigates the broader applicability of method-level indexing in supporting software maintenance and evolution tasks. It examines how detailed insights into code relationships can aid developers in managing and evolving their projects more efficiently and effectively.
RQ4: What are the scalability and performance implications of implementing SearchSECO on a global scale?
Justification: This question assesses the practicality of deploying SearchSECO across the vast global software ecosystem. It examines whether the system can handle large-scale data efficiently, ensuring that it remains useful and effective in real-world scenarios.

With these research questions, we tackle the broader challenge of what it means to look at the SECO on a global level through source code. It makes the project ambitious, although this is a recurring theme in the repository mining community.

3. Related Work

Seulbae et al. introduced one state-of-the-art solution named VUDDY (VUlnerable coDe clone DiscoverY) project which is a scalable approach for code clone detection [5]. This work identifies the code clone and vulnerability by leveraging the syntactic and symbolic information of the code. This research worked on four types of code clones that have been recognized and published by scientific papers according to the granularity units such as token level, line level, function level, file level, etc. However, this work has limitations in terms of accuracy and consistency because of granularity abstraction. This leads to higher false negatives and the paper also acknowledges that trustworthiness a concern originates from the false negative. We effectively use similar techniques as VUDDY, however, we are storing all the methods that we encounter instead of only the methods encountered with potential vulnerabilities. This for instance enables us to study license violations, something that VUDDY was not built for.

License conflicts and violations are universal issues, and software licenses generally fall into two categories. The first one is declared licenses which is specified for the whole project and the second one is in-code licenses which are directly attached to files throughout the entire directory tree [8]. Most of the violations originated from the declared to-in-code mismatches on the other hand declared to declared-to-declared inconsistencies also occur but less often.

Research shows [9] there are multiple reasons behind code license violations in open-source code software (OSS). The first one originates from the resource and time constraints of software developers and they do not want to focus on trivial tasks. And the second one is related to the misconceptions about the nature and characteristics of open-source licenses which are escalating chronologically because of the large number of repositories produced. Another paper mentioned about the incompatibility of licenses among components, for example, GNU General Public License (GPL) has multiple versions but it doesn't have backward compatibility such as components released under GPL version 3 are not compatible with components released under GPL version 2 [10]. However, when the same project shares two different licenses then it's an inconsistency but that does not necessarily mean a license conflict. License conflict means contradictory, incompatible obligations or contradictory rights [8].

Wolter et al. also published a research work based on 1,000 GitHub repositories and the found that fifty percent of the work repositories did not include a complete and accurate list of all the licenses associated with the code. However, out of these 10% has a mismatch between permissive and copyleft licenses. This work heavily relied on existing open source tools such as Nomos, ScanCode etc. also mentioned the necessity of license scanning tools directly from the code.

The FOSSology project is one of the widely used projects for license detection [11]. Based on regular expressions a license scanner has been developed and the main tool of FOSSology. For the purpose of license scanning, FOSSology introduced open-source license scanners, for example, Ninka. Within source code comments, Nika can analyze sentences and it can recognize over 120 different licenses [12].

Some research done based on binary code clone detection for detecting software code released in binary form [13] which mentioned that upstream suppliers often provide solutions in binary form, thus it's difficult to assess the existence of unlicensed third-party code. It's also mentioned that the license violations are not accidental, but rather more systematic, and for most software and hardware products this is a large-scale problem.

This research is based on a previously proposed solution named SearchSECO by Slinger Jansen et al. for a hash-based index that aims to collect billions of open-source files from open-source repositories to provide full software provenance which also address the license conflicts and violations problem [3].

4. Research Method

Software engineering is a maturing field, and repository mining as a branch of it, is as well. This can be observed especially when looking at the way in which empirical software engineering research is being conducted and evaluated. In this thesis project, we follow the empirical software engineering research

standards for our sub-projects.[1]

To address the issues and problems of vulnerability detection, license conflicts, software maintenance, and scalability in large-scale software ecosystems, this PhD research will enhance the capability and explore the possibility of an already proposed solution named SearchSECO [3]. This PhD research will be under the Design Science Research (DSR) paradigm because this research is based on software artifact development which involves an evaluation process for code license violation detection. The evaluation process will have two steps: the first step is a vulnerability benchmarking framework that will be established to evaluate the capabilities of the SearchSECO software artifact. The second step is a case study in an industry organization and for this purpose, we plan to incorporate other research methods under DSR to answer a particular research question. For example, to deploy SearchSECO in an industry organization and to identify code license violations in large unidentified code bases we will follow Action Design Research (ADR) as a part of this PhD research (section 6, WP2 and WP3) [14, 15]. ADR is particularly designed to develop, work, and evaluate with organizational settings where researcher intervention is expected [16]. This method centers on creating, intervening, and evaluating an artifact that embodies the researchers' theoretical foundations and intentions while incorporating user influence and the impact of real-world use.

A validity threat for our project is that we can hardly generalize our findings to the whole software ecosystem. Even though we currently have analyzed the top 100.000 projects on Github, that is still only a fraction of the total Github source code, let alone the worldwide software ecosystems' source code. For now, we will refrain from making inductive statements outside of the scope of our own data set.

A construct validity threat is on the definition of a clone. Currently, we are storing any clone, including for instance, getters and setters, in an abstract manner. However, as these are typically uniform and we use a high level of abstraction, we find many false positives in our clone set. In the near future we hope to counter this validity threat by setting a standard length for a 'valid' clone, e.g., a minimum of five lines of code.

A proposed and partially implemented system diagram is shown in figure 1. It illustrates the project SearchSECO intended to collect source code from the worldwide software ecosystem and store method-level code with the call graph of the code in a Software Method Knowledge Base (SMKB), which allows for structural analysis of the source code. For example, license violation and vulnerability patterns will be identifiable utilizing the call graph of a source code. Thus, in the software engineering domain more specifically in repository mining SearchSECO is a radical innovation [3]. To construct SeacrhSECO we follow four lines of query: first, we develop parsing techniques and design work distribution mechanisms to explore the global SECO. Next, we store the collected methods in the SecureSECO Knowledge Base (SMKB). Third, we apply basic data analysis techniques to the stored data within the SMKB. Finally, we leverage artificial intelligence to perform graph mining on the worldwide SECO graph for deeper insights [3].

5. Implementation Plan and Timeline

A total of four years PhD thesis plans has been added to the Table1. The first paper with the initial results has been accepted for publication at BENEVOL 2024 conference. The title is "Work in Progress Paper: Detecting Method Level License Conflicts in the Worldwide Software Ecosystem". This paper demonstrated code-level license extraction and violation detection as a definitive method for ensuring license compliance in borrowed code, independent of any declarations. Using SearchSECO, we examined 3,500 repositories from leading software companies to assess the prevalence of violations. Our analysis uncovered approximately 32,000 violations in total.

[1]For more details on empirical software engineering standards, see Empirical Standards Repository.

[1]This diagram provides an overview of the SearchSECO system components and their interactions for efficient code fragment search at the method level.

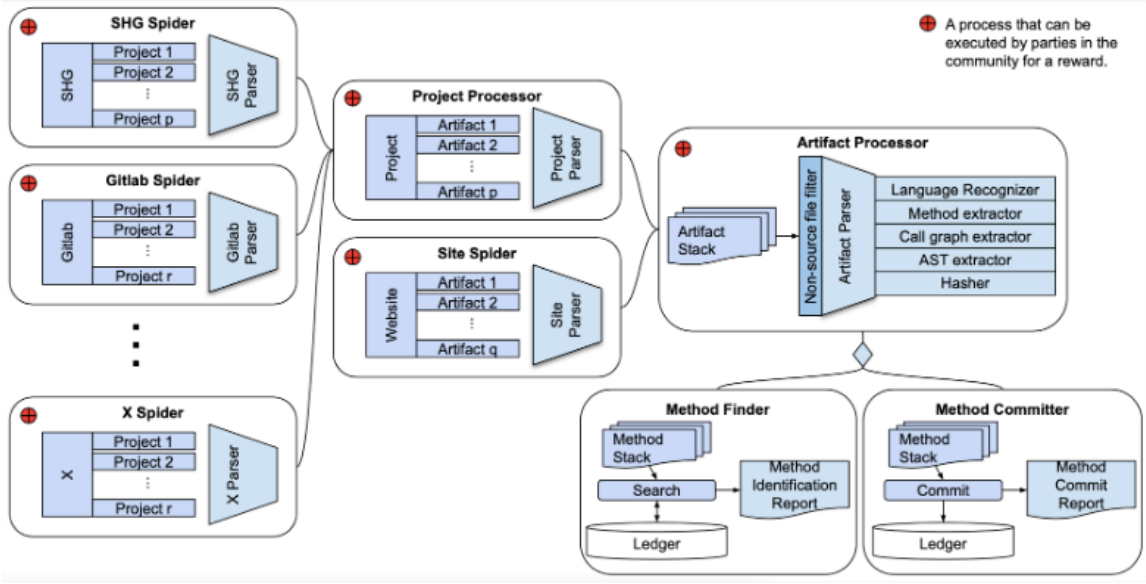

Figure 1: SearchSECO System Architecture [3]

6. Future work

The FOSSology project first introduced an ML-based solution for license identification problems utilizing license classification [17]. Another research introduced Machine learning-based license exception detection [18]. However, only a few ML-based solutions are introduced, and some topics such as license violation are not explored. Moming et al. introduced an ML-based solution named ModelGo for license conflict detection [19]. Research needs to be done on this problem. In our SearchSECO project introducing ML-based license violation solution could be an interesting experiment for the scientific and research community. However, another idea is to create a hybrid solution, utilizing the current proposed SearchSECO license violation and combining an ML-based solution.

A recent development is the use of LLMs for generating code. Frequently, the code that is generated follows exact patterns from licensed code, meaning that the LLM could potentially be offering licensed code, thereby stimulating license violations by the software engineer [20]. While we cannot guarantee it currently, we look forward to exploring whether we can identify such licensed code suggestions, with the goal of improving these LLMs to avoid licensed code.

Incorporating existing tools such as Binary Analysis Tool (BAT) [13] binary code clone detection can be done in the SearchSECO project too. Utilizing bytecode scanning tools like JEB Decompiler or ProGuard [21] that can extract relevant license details directly from compiled code (e.g., Java .class files or .NET assemblies). Bytecode-level analysis allows for the detection of licensing information even when source code is unavailable [22].

7. Contribution of the Thesis

SearchSECO has the aim of creating a new sense of provenance in software engineering, where we try to find the earliest version of a code clone and its authors. Provenance, which concerns the origins of an artifact, has been neglected in software engineering for far too long. For instance, if we look at the Dieselgate scandal, the code which violated the tests was never found. With SearchSECO, it would perhaps have been possible to identify the intended code [23].

As we are performing design science, with potentially highly useful artifacts, both for research and industry, we foresee several routes towards research impact. For one, we hope to apply and evaluate our artifacts in case studies. Secondly, if the technology proves valuable for the industry, we could consider spinning out the SearchSECO features into a startup.

Table 1
SearchSECO Development and Deployment Plan Over Four Years

Year	WP(s)	Objective	Milestones
Year 1	WP0 & WP1	Understanding SearchSECO and Code Borrowing, Using SearchSECO for determining license conflicts	Q1: Finalize the enhancements to SearchSECO for license conflict detection. Q2: Apply prototypes to real-world datasets and gather preliminary results. Q3: Conduct initial tests and validations of the developed artifacts and publish the code license violation detection paper. Q4: Enhance the SearchSECO license-checking capability for benchmarking.
Year 2	WP2	Benchmarking SearchSECO's capabilities for identifying vulnerabilities	Q1: Develop the vulnerability benchmarking framework and tools and conduct detailed case studies on license conflict detection and vulnerability benchmarking. Q2: Refine the artifacts based on feedback and testing results. Q3: Publish initial findings in peer-reviewed venues. Q4: Expand the dataset and improve the scalability and robustness of the tools.
Year 3	WP3	Deploying SearchSECO in an industry organization: identifying code in large unidentified code bases	Q1: Analyze scalability and performance implications of implementing SearchSECO on a global scale. Q2: Deploy code in an industry organization and collect data. Q3: Write articles for the scalability and performance implications of implementing SearchSECO on a global scale and publish papers in journals and conferences. Q4: Conduct extensive evaluations and final validation of the developed artifacts.
Year 4	WP4	StackOverflow Data Analysis with SearchSECO	Q1: Collect StackOverflow data and check against SearchSECO database. Q2: Publish comprehensive results and methodology in high-impact journals and conferences. Q3: Disseminate the tools and methodologies to the broader community and present the technical artifact and share knowledge with the community. Q4: Complete the dissertation writing and defense.

Acknowledgments

We wish to thank Geert-Jan Giezeman, Wouter Beffers, and Deekshitha for their important contributions to the SearchSECO project on Github. This research was funded by the Business Finland project 6G Bridge – 6G software for extremely distributed and heterogeneous massive networks of connected devices (8516/31/2022).

Declaration on Generative AI

ChatGPT 4o was used for polishing the grammar and spelling of the text in this document.

References

[1] S. Jansen, P. Buxmann, T. Kude, K. Popp, Proceedings of European Workshop on Software Ecosystems: 2012 - Walldorf, Synomic Academy, Books on Demand, 2013. URL: https://books.google.fi/books?id=BbivKcE6vWMC.

[2] S. Jansen, M. A. Cusumano, S. Brinkkemper, Software ecosystems: analyzing and managing business networks in the software industry, Edward Elgar Publishing, 2013.

[3] S. Jansen, S. Farshidi, G. Gousios, J. Visser, T. van der Storm, M. Bruntink, Searchseco: A worldwide index of the open source software ecosystem, in: The 19th Belgium-Netherlands Software Evolution Workshop, BENEVOL 202, CEUR-WS. org, 2020.
[4] K. K. Chaturvedi, V. Sing, P. Singh, Tools in mining software repositories, in: 2013 13th International Conference on Computational Science and Its Applications, IEEE, 2013, pp. 89–98.
[5] S. Kim, S. Woo, H. Lee, H. Oh, Vuddy: A scalable approach for vulnerable code clone discovery, in: 2017 IEEE symposium on security and privacy (SP), IEEE, 2017, pp. 595–614.
[6] Secureseco, 2024. URL: https://github.com/SecureSECODAO/searchSECO-miner, [www document], [Accessed on 12.09.2024].
[7] Searchseco, 2024. URL: https://secureseco.org/secureseco-introduction/searchseco/, [www document], [Accessed on 12.09.2024].
[8] T. Wolter, A. Barcomb, D. Riehle, N. Harutyunyan, Open source license inconsistencies on github, ACM Trans. Softw. Eng. Methodol. 32 (2023). doi:10.1145/3571852.
[9] Y. Golubev, M. Eliseeva, N. Povarov, T. Bryksin, A study of potential code borrowing and license violations in java projects on github, in: Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 54–64. doi:10.1145/3379597.3387455.
[10] A. Mathur, H. Choudhary, P. Vashist, W. Thies, S. Thilagam, An empirical study of license violations in open source projects, in: 2012 35th Annual IEEE Software Engineering Workshop, 2012, pp. 168–176. doi:10.1109/SEW.2012.24.
[11] R. Gobeille, The fossology project, in: Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 47–50. doi:10.1145/1370750.1370763.
[12] M. C. Jaeger, O. Fendt, R. Gobeille, M. Huber, J. Najjar, K. Stewart, S. Weber, A. Wurl, The fossology project: 10 years of license scanning, IFOSS L. Rev. 9 (2017) 9.
[13] A. Hemel, K. T. Kalleberg, R. Vermaas, E. Dolstra, Finding software license violations through binary code clone detection, in: Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 63–72. doi:10.1145/1985441.1985453.
[14] Engineering research (aka design science), 2024. URL: https://github.com/acmsigsoft/EmpiricalStandards/blob/master/docs/standards/EngineeringResearch.md, [www document], [Accessed on 03.10.2024].
[15] R. Baskerville, J. Pries-Heje, J. Venable, Soft design science methodology, in: Proceedings of the 4th international conference on design science research in information systems and technology, 2009, pp. 1–11.
[16] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, R. Lindgren, Action design research, MIS quarterly (2011) 37–56.
[17] R. Gobeille, The fossology project, in: Proceedings of the 2008 international working conference on Mining software repositories, 2008, pp. 47–50.
[18] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, D. Poshyvanyk, Machine learning-based detection of open source license exceptions, in: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 118–129. doi:10.1109/ICSE.2017.19.
[19] M. Duan, Q. Li, B. He, Modelgo: A practical tool for machine learning license analysis, in: Proceedings of the ACM Web Conference 2024, WWW '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 1158–1169. doi:10.1145/3589334.3645520.
[20] V. Majdinasab, A. Nikanjam, F. Khomh, Trained without my consent: Detecting code inclusion in language models trained on code, 2024. URL: https://arxiv.org/abs/2402.09299.
[21] E. Lafortune, Shrink your java and android code, proguard., 2016. URL: https://www.guardsquare.com/proguard, [www document], [Accessed on 21.10.2024].
[22] J.-T. Chan, W. Yang, Advanced obfuscation techniques for java bytecode, Journal of Systems and Software 71 (2004) 1–10. URL: https://www.sciencedirect.com/science/article/pii/S0164121202000663. doi:https://doi.org/10.1016/S0164-1212(02)00066-3.

[23] O. Boiral, M.-C. Brotherton, A. Yuriev, D. Talbot, Through the smokescreen of the dieselgate disclosure: Neutralizing the impacts of a major sustainability scandal, Organization & Environment 35 (2022) 175–201. doi:10.1177/10860266211043561.