SearchSECO: A Worldwide Index of the Open Source Software Ecosystem

Slinger Jansen
Utrecht University, The Netherlands
Lappeenranta University, Finland
slinger.jansen@uu.nl

Siamak Farshidi
University of Amsterdam, The Netherlands
Utrecht University, The Netherlands
s.farshidi@uva.nl

Georgios Gousios
Technical University Delft
The Netherlands
g.gousios@tudelft.nl

Tijs van der Storm
University of Groningen, The Netherlands
CWI, The Netherlands
storm@cwi.nl

Joost Visser
Leiden University
The Netherlands
j.m.w.visser@liacs.leidenuniv.nl

Magiel Bruntink
Software Improvement Group
The Netherlands
m.bruntink@sig.eu

***Abstract*—Repository mining research is a data-intensive domain with a focus on source code. There are many ways to search for code in the worldwide software ecosystem, but these search methods are inefficient and only cover small parts of the software ecosystem. One of the problems is granularity: it is possible to search through code on a file-level and cover a significant part of the software ecosystem or search for a line of code and only cover a small part of the software ecosystem, but not both.**

We propose SearchSECO: a language-agnostic search engine and research platform that searches through abstract representations of source code methods. We use SearchSECO to search across the worldwide software ecosystem and index the encountered methods. With SearchSECO, the field is advanced because it (1) provides finer-grained and more efficient searches, (2) covers more of the software ecosystem than other search mechanisms, and (3) provides mechanisms for source code provenance.

I. Introduction

Reuse has been a blessing and a curse for modern software engineering. While frequent replication of successful fragments of code leads to initial high productivity, it may also be one of the lead causes for technical debt, replication of bugs and vulnerabilities, and poorly maintainable code. Fortunately, through research, we have found many ways to identify code clones and re-engineer them [1].

A software ecosystem (SECO) is a set of actors functioning as a unit and interacting with a shared market for software and services, together with their relationships [2]. Society is entirely dependent on a healthy worldwide SECO, as every aspect of our society is dependent on software. We propose a software provenance theory, where the provenance of each software engineering artifact is known in the entire software supply network. With software provenance, we can provide guarantees about software artifacts' quality, introducing a novel layer of trust in the worldwide SECO.

Methods are typically a few code lines that express a set of operations on an object, are contained in larger code files, and correspond with the way a software object is conceptualized by software engineers. This project aims to collect the source code in the worldwide SECO and store it at the method level, including the call graph of the code, in a Software Method Knowledge Base (SMKB). This affords us to do a structural analysis of the source code, for instance, to identify vulnerability patterns in call graphs. SearchSECO makes the worldwide SECO searchable at a finer level of granularity than was possible before. This is a radical innovation in software engineering research and, more specifically, repository mining [3], as it enables high-performance worldwide code searches that cross-cut language ecosystems [4].

This project is **ambitious**: we want to analyze more than 8 billion open-source files[1] and extract even more code fragments from it [5]. The project is **radical**: we redesign trust in the worldwide SECO through a theory of software provenance, and if completed, software engineering will never be the same again. But most importantly, the project has **impact** on society: we provide full software provenance, thereby enabling software producers to guarantee who touched the software, and consumers knowing precisely where their software came from, that it was ethically produced, and how vulnerable it is in real-time.

II. Mapping the Worldwide Software Ecosystem

We can map the full SECO with SearchSECO, as dependencies between projects also need to be stored explicitly. With these dependencies, it becomes possible to identify "rotten" branches in the dependency tree, i.e., where one required package high up in the dependency tree is determined vulnerable. From that point onward, all packages that depend on the vulnerable package can also be considered compromised.

With the worldwide SECO mapped, we can perform all kinds of network analysis, such as social analysis (who works with whom? Who copies code from whom?), technical

[1]Currently, the software heritage graph contains 8 billion source code files.

		SearchSECO	GHtorrent	SHG	libraries.io	SearchCode	FASTEN	GL & GH
General Properties	**Funding (Proprietary/Public/Community)**	Co	Pu	Co	Pr	Pr	Pu	Pr
	Parses code	Y	N	N	Y	N	Y	N
	Works in a distributed manner	Y	Y	N	N	N	N	N
Source code level	**Search source code lines**	N	Y	Y	N	Y	N	N
	Search abstract syntax tree	Y	N	N	N	N	N	N
Method level	**Search call graph**	Y	N	N	N	N	Y	N
	Search methods by hash	Y	N	N	N	N	N	N
	Search method meta-data	Y	N	N	N	N	N	N
Author relationships	**File authorship**	Y	Y	N	N	N	N	Y
	Method authorship	Y	N	N	N	N	N	N
Project/Package level	**Project Information**	Y	Y	Y	Y	N	Y	Y
	Monitors package releases	Y	N	N	Y	N	Y	Y
	Package dependency tree	Y	N	N	Y	N	Y	N
	Licensing information	Y	N	N	N	N	Y	Y

TABLE I

No other project performs abstract syntax tree code searches across source code methods, while that is urgently needed for RSEs. SearchSECO also stores more meta-data than others. SHG is the Software Heritage Graph. GL and GH are Gitlab and Github

analysis (what is the essential package worldwide?), and bus-factor analysis (which crucial packages have very few software engineers?). We can support open-source software engineers with further research about:

- **Relationships between methods.**
  - *Study method co-evolution across projects* - At present, it is impossible to track methods that have been copied and co-evolved reliably. With SearchSECO, empirical software engineering researchers track method fragments evolution over time.
  - *Weaknesses tracked, fixes propagated* - With vulnerability databases such as Google's https://www.vulncode-db.com/, it becomes possible to rapidly identify syntactically equivalent method fragments that contain vulnerabilities [6].
- **Relationships between authors.**
  - *Fine grained authorship* - Using commit data from the different repositories, we store meta-data about method authorship. For instance, this enables us only to notify authors who have worked on a particular method and not all authors who have edited a particular code file.
  - *Copy-paste behavior* - It is almost impossible to assess which authors copy their methods from where. With SearchSECO, it is possible to identify which methods are copied from StackOverflow.com [7].
- **Relationships between software projects.**
  - *Establish package dependencies and cohesion* - we analyze dependencies between packages and assess how strongly two components are coupled. Combining and analyzing these data makes it possible to create trust models around projects, which indicate the trustworthiness of packages and projects and make it easier for engineers to select packages in the SECO.
  - *License violations* - Methods are stored with their license information, enabling license violation detection.

A final interesting outcome is that it becomes possible to identify code fragments on StackOverflow.com and how they are copied into existing projects, similar to other works [8], [9]. It even becomes possible to notify software engineers who have copied fragments from StackOverflow.com that have been updated or fixed.

We expect the SearchSECO database to be instrumental in identifying new vulnerabilities and malware in software code. We aim for the SMKB to identify new source code problems, thereby contributing to (insight into) a safer SECO. We use several techniques for achieving this objective. First, we use machine learning techniques to predict code mutations of source code fragments to search for mutated code clones. Secondly, we use pattern matching techniques to explore software call graphs to search for vulnerabilities and malware, as many malware and viruses follow similar call graph patterns [10]. If vulnerabilities and malware are detected, we use the SearchSECO platform to notify the producers of packages and users of these packages through a notification system.

III. Realization

A workflow blueprint is provided in Figure 1, where it is illustrated how new data sources can be explored and processed to become part of the SMKB. The jobs that are to be done, such as regular synchronization with Gitlab and the Software Heritage Graph, can be distributed across the community members. In this way, the community updates and maintains the SMKB.

We follow four lines of inquiry to construct SearchSECO. First, we develop parsing techniques and work distribution mechanisms for exploring the worldwide SECO. Secondly, we store the methods in the SMKB. Thirdly, we analyze the data in the SMKB using basic data analysis techniques. Fourthly,

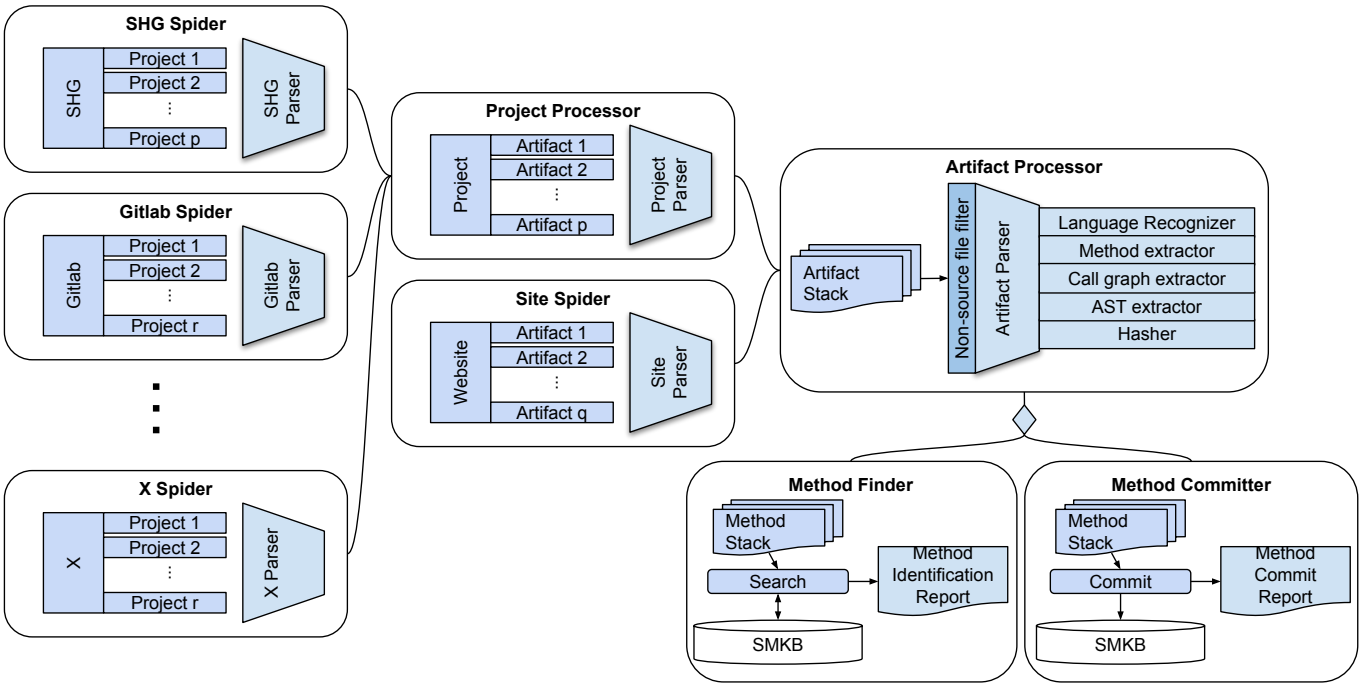

Fig. 1. *The extraction process follows a three-step process. First, projects are identified on different project forums, such as the Software Heritage Graph, Gitlab, and other project repositories. Different artifacts are extracted from web sites such as stackoverflow.com. Secondly, these projects are parsed into artifacts from which fragments are extracted, including their out-calls to other methods. Thirdly, this information is stored in the SMKB.*

we employ artificial intelligence to perform graph mining on the worldwide SECO graph.

A. Parsing Worldwide Software Ecosystem

We create a parser infrastructure that rapidly extracts methods and calls graphs from source code. Currently, we have parsers available for Java, C, and C++, using technology from the related FASTEN project[2], through srcML (https://srcml.org), and possibly also using Rascal [11].

First, we perform *language parametric clone detection*. Vuddy [12] is a tool for high-performance clone-detection for C. Using generic language technology, we will extend Vuddy to support other languages in the SECO. Vuddy operates at the function level; using the metaprogramming system Rascal we will develop a generic front-end to Vuddy based on existing and new parsers to support other languages than C/C++.

Secondly, we *develop generic models for cross-language dependencies.* To track dependencies induced by call-graphs and other relations (e.g., inheritance), we will develop generic, cross-language models to represent such dependencies. Rascal already supports the extensible M3 model [13], for single-language source projects. We will extend this to support modeling source code facts across different programming language SECOs, as well as the representation of metadata such as authorship, provenance, and versioning.

Thirdly, we use *AI-assisted development of robust, extraction-oriented parsers*. We require parsers for programming languages that might not yet exist. For instance, Rascal's current standard library includes robust parser front-ends for Java and C/C++. However, developing parsers for full programming languages requires significant effort. We will investigate new AI-based techniques to (semi-)automatically derive parsers using a combination of grammar inference techniques and corpus analysis. These parsers might not be accurate enough for developing a compiler but will be sufficiently fine-grained to extract function bodies and identify call sites.

Fourthly, we aim to do *"Diff"-based parsing and extraction*. The volume of code in the SECO that needs to be analyzed is huge and continuously evolving. Altogether parsing and analyzing the code of software projects from scratch will not scale. To have repository spiders monitor and analyze repositories continuously, we will develop techniques for "diff"-based parsing and extraction. Instead of parsing/analyzing full source files, these techniques will analyze the difference between versions of files (e.g., as derived from Github) and incrementally update the SMKB.

B. Storage and Retrieval of Worldwide Code Clones

We start by *exploring the different manners in which code clones can be extracted from a software project*. We are inspired by Vuddy [12], a tool that extracts methods from C and C++, extracts their Abstract Syntax Tree (AST), and hashes this tree. It has excellent performance for larger codebases, such as the one proposed in SearchSECO. We want to explore whether methods are the ideal code clones or whether we can use alternatives, such as groupings of variables in a method. Also, we want to explore whether other objects also need to

[2]https://www.fasten-project.eu/

be hashed, such as headers in header files and comments. Vuddy is available as open-source as a tool called hmark (https://github.com/iotcube/hmark).

We aim for *clones to be "addressable" through an abstract representation*, such as their AST, but also through meta-data that is stored about them, such as their project title, method signature, and version release. These data enable us to address the same fragment in different versions to identify whether a particular vulnerability is present in multiple versions of the same package.

We need to *rapidly retrieve clones from our extensive index of code clones*, which we envision to store at least several billions of code clones. We identify hashing techniques that best suit this purpose. Furthermore, we can index commercial software without its actual source code, leaving the organizations' premises by only storing hashes.

We use *machine learning to predict how a code clone may mutate in the future*. By generating such data, we can develop a technique that stores hashes of those mutations as well, to identify more code clones [14]. While this is highly experimental, we believe we may be able to catch more vulnerabilities this way.

C. Collect Source Code for Analysis

We collect source code for analysis from existing source code databases. We need the database to be filled with code fragments and trust data to prove that our proof of concept works. We plan to use the multitude of existing repositories with archival software knowledge, such as the Software Heritage Graph [15] and GHTorrent [16] and use similar techniques for de-duplication of the massive amounts of data from those platforms. However, this is a big challenge: the Software Heritage Graph, one of the largest open-source code databases, currently contains 8 billion code files, including at least as many methods. We take a distributed approach to develop a job scheduler and a client that different teams in the project can run to perform the data collection and analysis.

A *job scheduler is developed that maintains its list of jobs to be done*. Worker nodes can pick up these jobs in the ecosystem. This work will be based on the CrossFlow infrastructure [17], as developed in the CrossMiner project. Examples of automated tasks are:

1) Spider an existing project repository for updates
2) Extract code fragments from Stackoverflow
3) Parse a new project and identify the languages used
4) Send out alerts to owners of encountered code fragments
5) Check whether evidence of a code fragment still exists

These automated tasks will be incentivized to ensure positive contributions to the community. Some jobs may stay in the scheduler for a long time; as for these jobs, the correct parser may not yet be available and still needs to be developed. In this way, we can easily prioritize which parser needs to be most urgently built.

Participants that use the ecosystem use it to search the SMKB for code fragments. However, new projects need to be parsed, extracted, and stored in the SMKB as well. Furthermore, unique language ecosystems need to be added, and large project repositories need to be mined. Incentivized participants in the ecosystem must perform these tasks. For this, worker nodes must be developed that can perform automated tasks.

Once the clients are ready, *the data collection can begin, and the database can be filled*. The industry stakeholders will want to see that the database is working, so we want to use it first. We will start by analyzing several large projects (Linux, for instance) so that we are ensured to have relevant and urgently needed data. We aim to lean on the repository mining and BENEVOL community for spreading the workload across different researchers. Throughout the project, we will collect data about software engineers in the SECO. However, as SearchSECO does not make the SMKB a surveillance instrument, we must use design principles that do not easily link software engineers to their identity information.

D. AI Assisted Graph Mining for Vulnerabilities

We use graph mining and other techniques for classifying vulnerabilities from vulnerability databases, to identify new vulnerabilities in existing and newly added code. There are two parts to this. First, we want to structure known vulnerabilities so that the vulnerabilities and permutations of such vulnerabilities can best be found in our code database. Secondly, we want to establish ways to automatically propose fixes. Using existing vulnerability codebases such as VulnCode (https://www.vulncode-db.com/), we will *try to detect existing vulnerabilities in source code and alert the owners* of the code about the vulnerability. Other projects have demonstrated that this approach is successful in detecting vulnerabilities [18].

Using the FASTEN project call graph (ecosystem-wide caller-callee relationships), we can *pinpoint vulnerabilities at the method level* and explore their propagation across their ecosystem. This enables pattern-based graph searches that can be used to detect malware [19].

The severity of vulnerabilities in code depends on several factors. If the code has been fixed in a newer release, its severity is relatively low. However, if any other packages depend on this code, for instance, using dependency data from the FASTEN project and its vulnerability, we can warn the creators about the vulnerability.

As the fixes in vulnerability databases are typically well structured and relatively easy to fix, *we could automatically generate pull requests for the code to be fixed.* Furthermore, if the tooling developed in this project is adopted widely, we could warn about vulnerable code at the time of a commit.

IV. Conclusion

We proudly present SearchSECO to index all software methods in the worldwide SECO. With SearchSECO, we aim to serve future empirical software engineering researchers with new tools and data to conquer new research challenges. Furthermore, we hope that our innovations can lead to a safer and more secure worldwide SECO. With this paper, we hope to gather feedback and ideas from the BENEVOL community to implement and improve SearchSECO.

ACKNOWLEDGMENT

The SearchSECO project, which is part of the SecureSECO initiative, is currently supported by **Gitlab**, **the Software Improvement Group (SIG)**, the **Lisk Center**, **Cisco**, **Centric**, and **KPN Security**. See SecureSECO.org/partners for more information. SecureSECO is endorsed by the **Secure Software Alliance**, the **Vereniging Software Engineering Nederland (Versen.nl)**, and the **Blockchain Coalition**. We are also in contact with the **eSciences center**, **SurfSARA**, and **DANS**, whom we see as strategic partners in designing our computing and data storage resources.

We would like to thank all the SecureSECO student team members for their ideas in this paper: Venja Beck, Floris Jansen, Fang Hou, Elena Baninemeh, Luuk van Driel, Jozef Siu, Swayam Shah, Donny Groeneveld, and Tom Peirs.

CONTRIBUTION STATEMENT

This paper was composed by Slinger Jansen, who has brought the research teams together to create the concepts behind SearchSECO. Siamak Farshidi has assisted in writing the article, creating Table I, and proof reading the work. Tijs van der Storm, Georgios Gousios, Magiel Bruntink, and Joost Visser were, in no particular order, responsible for the subsections in the Realization Section. Furthermore, they have contributed greatly to the development of the SearchSECO project and the SecureSECO initiative.

REFERENCES

[1] M. Bruntink, A. Van Deursen, R. Van Engelen, and T. Tourwe, "On the use of clone detection for identifying crosscutting concern code," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 804–818, 2005.

[2] S. Jansen, M. A. Cusumano, and S. Brinkkemper, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar Publishing, 2013.

[3] T. Siddiqui and A. Ahmad, "Data mining tools and techniques for mining software repositories: A systematic review," in *Big Data Analytics*. Springer, 2018, pp. 717–726.

[4] A. Serban, M. Bruntink, and J. Visser, "Graphrepo: Fast exploration in software repository mining," *Arxiv 2008.04884*, 2020.

[5] G. Rousseau, R. Di Cosmo, and S. Zacchiroli, "Software provenance tracking at the scale of public source code," *Empirical Software Engineering*, 2020.

[6] J. Bader, A. Scott, M. Pradel, and S. Chandra, "Getafix: learning to fix bugs automatically," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–27, 2019.

[7] L. An, O. Mlouki, F. Khomh, and G. Antoniol, "Stack overflow: A code laundering platform?" in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, pp. 283–293.

[8] R. Abdalkareem, E. Shihab, and J. Rilling, "On code reuse from stackoverflow: An exploratory study on android apps," *Information and Software Technology*, vol. 88, pp. 148–158, 2017.

[9] S. Baltes and S. Diehl, "Usage and attribution of stack overflow code snippets in github projects," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1259–1295, 2019.

[10] T. Wüchner, A. Cisłak, M. Ochoa, and A. Pretschner, "Leveraging compression-based graph mining for behavior-based malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 99–112, 2017.

[11] P. Klint, T. van der Storm, and J. J. Vinju, "Rascal: A domain specific language for source code analysis and manipulation," in *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE Computer Society, 2009, pp. 168–177.

[12] S. Kim, S. Woo, H. Lee, and 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.

[13] B. Basten, M. Hills, P. Klint, D. Landman, A. Shahi, M. Steindorfer, and J. Vinju, "M^3: a General Model for Code Analytics in Rascal," in *Proceedings of the first International Workshop on Software Analytics, SWAN*, 2015.

[14] C. Ragkhitwetsagul and J. Krinke, "Siamese: scalable and incremental code clone search via multiple code representations," *Empirical Software Engineering*, pp. 1–49, 2019.

[15] A. Pietri, D. Spinellis, and S. Zacchiroli, "The software heritage graph dataset: Public software development under one roof," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19. IEEE Press, 2019, pp. 138–142.

[16] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[17] D. Kolovos, P. Neubauer, K. Barmpis, N. Matragkas, and R. Paige, "Crossflow: a framework for distributed mining of software repositories," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 155–159.

[18] N. H. Pham, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Detection of recurring software vulnerabilities," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 447–456.

[19] X. Hu, T.-c. Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 611–620.