Poster: An In-Depth Analysis of the CodeQL Ecosystem

Jean-Charles Noirot Ferrand
School of Computer, Data & Information Sciences
University of Wisconsin-Madison
Email: jcnf@cs.wisc.edu

Patrick McDaniel
School of Computer, Data & Information Sciences
University of Wisconsin-Madison
Email: mcdaniel@cs.wisc.edu

***Abstract*—Open-source software relies on automated static analysis tools to prevent the introduction of vulnerabilities in code. In particular, GitHub's code query engine, CodeQL, is among the most used static analysis tools. However, given its breadth (multiple languages) and depth (query engine), it is hard to quantify the overall health of the tool and its ecosystem. In this work, we perform a measurement study on the CodeQL ecosystem across three interconnected angles: the queries and their properties, the community, and the development. We first study the ecosystem from publicly available information (GitHub pull requests, git history, changelogs, etc.). Then, we measure the practical impact of those changes on vulnerability detection by running the different version of the tools on known vulnerable software from prior CVEs. We find that more than 50 CVEs could have been prevented if CodeQL was used.**

1. Introduction

Open-source software, as part of an ever-growing supply chain, is prone to attacks [1]. Recent events such as the XZ backdoor or Log4shell [2] illustrate the existence and importance of security incidents within the supply chain. To eliminate as many attack vectors as possible, software projects undergo automated static analysis (through Github CI/CD pipelines for instance), therein relying on the tools developed to this end: static application security testing (SAST) tools. Among these tools is one of GitHub's Advanced Security product, CodeQL: a semantic code analysis engine with predefined suites of queries. Since its acquisition and introduction in 2019 by GitHub, CodeQL has garnered attention by developers [3] and academics [4] through its depth of analysis and its breadth of coverage. Indeed, in 2022, GitHub reported storing CodeQL databases for more than 200 000 repositories [3].

While prior work has studied SAST tools (including CodeQL) through both quantitative and qualitative lenses, such evaluations often prioritize breadth of tools over depth as they study multiple tools at the performance level (i.e. accuracy, precision, etc.) or through a qualitative analysis (e.g. user study). Further, these performance evaluations are often limited to a small scope of CWEs and/or programming languages (often C/C++ or Java). Therefore, the space currently lacks comprehensive analyses on such tools to understand their pitfalls and advantages. Such analyses are difficult to do as most SAST products are not open-source, leaving in-depth measurements limited and incomplete. Given its fully open-source and community-driven nature, CodeQL presents itself as a great opportunity to measure and understand how a security static analysis tool is developed and updated.

In this work, we posit that the health of the CodeQL ecosystem is measurable, therein enabling an accurate assessment of the tool's performance and development properties. We consider three axes of the ecosystem: the rules, the community, and the development. First, we derive and measure several properties of rules such as their accuracy, their sophistication (amount of scaffolding they are based on), and their lifecycle. Then, we characterize the community surrounding CodeQL, with a focus on the contributors, maintainers, and their organization(s). Finally, we take a closer look at the development, analyzing at a finer granularity each change and its impact on the performance of the tool in practice.

We apply the security suite of CodeQL on CVEFixes [5]—a dataset of CVEs and their fixing commits. Our analysis spans more than 7 000 CVEs for 7 languages and 10 versions of CodeQL (between v2.6.0 and v2.24.0), resulting in a total of more than 140 000 applications of the tool.

2. Background

CodeQL. CodeQL—the successor to LGTM following the acquisition of Semmle by GitHub in 2019—is an open-source static analysis tool that is part of the GitHub Advanced Security suite. It is used by developers as a way to secure their code [6] and by academics for evaluation [7] or focused studies [8] as it enables to create specific queries that leverage the tool's code extraction capabilities. It is powered by the .QL language created by Semmle in 2007, a declarative and object-oriented language to retrieve information from relational databases. Typically, a CodeQL analysis involves three components: an extractor (mapping the source code to a queryable database), a set of queries, and their underlying libraries.

Usage in practice. While it is possible to use CodeQL manually (i.e. through the CLI), most projects use

it as part of their CI/CD pipelines or using the dedicated VS Code extension. GitHub exposes the two GitHub Actions `github/codeql-action/init` and `github/codeql-action/analyze` to set up the environment for CodeQL and apply the analysis, respectively.

3. Methodology and Results

3.1. Data Sources

To make a comprehensive measurement of CodeQL, we employ diverse sources of information.

CodeQL. We use the official GitHub repository[1] (containing the implementation of the extractors, libraries, rules, and the changelogs) to characterize the development. To use the tool in practice, we use the official releases of the CLI.

CVEs and performance. As a way to measure the practical performance of the tool, we use CVEFixes [5], a dataset consisting of CVEs and corresponding commit fixes.

Papers. We gather papers by querying Google Scholar for any mention of "CodeQL" and classify them into four categories: mentions CodeQL, evaluates CodeQL, applies CodeQL, or improves upon it.

Users. To measure adoption of the tool, we use a dataset of GitHub Actions workflows [9] containing the full workflows history of over 3M+ workflows of 40k+ repositories.

3.2. Rules and Properties

To gauge the accuracy of rules, we run CodeQL on repositories that have a fix commit addressing a CVE from CVEFixes. Specifically, we run the tool on both the fix commit and the parent commit—which contains the vulnerability—and filter only the relevant findings following these properties: the findings must correspond to a similar CWE, it must be on a file of the fix, around the same line(s) as the fix, and not present in the fix (using `partialFingerprints` from the OASIS standard[2]).

Results. We observe little evolution of the accuracy across versions ($\leq$ 10% increase from v2.6.0 to v2.24.0). However, we find that more than 50 CVEs could have been found in time through CodeQL.

3.3. Community and Development

We measure the language expertise of CodeQL contributors by examining the git history of the CodeQL repository. In parallel, we consider the adoption rate of the tool depending on the language.

Results. We find that a majority of contributors do not contribute to a single language. Further, Java is the dominant expertise in the CodeQL ecosystem, which we attribute to the pre-existing work in static analysis for the language. However, this trend does not transfer to the adoption as we observed that Java is the third language with the most adoption (after Go and C#).

1. https://github.com/github/codeql/

2. https://docs.oasis-open.org/sarif/sarif/v2.1.0/cs01/sarif-v2.1.0-cs01.html#_Toc16012611

4. Conclusion

In this work, we perform a measurement study on CodeQL across three axes: rules, community, and development. We find that 50+ CVEs could have been detected prior to disclosure. We observe that contributor expertise concentrates in Java/C++, while adoption is led by Go and C#, hinting at a misalignment between developers and users.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2343611 and by PRISM, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*. Springer, 2020, pp. 23–43.

[2] National Vulnerability Database, "CVE-2021-44228 detail," https://nvd.nist.gov/vuln/detail/CVE-2021-44228, 2021.

[3] "CodeQL for VS Code: Download CodeQL databases from GitHub.com - GitHub Changelog," Sep. 2022.

[4] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," *Commun. ACM*, vol. 68, no. 2, p. 96–105, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3610721

[5] G. Bhandari, A. Brad, and O. Lopez, "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.

[6] G. Bennett, T. Hall, S. Counsell, E. Winter, and T. Shippey, "Do Developers Use Static Application Security Testing (SAST) Tools Straight Out of the Box? A large-scale Empirical Study," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Barcelona Spain: ACM, Oct. 2024, pp. 454–460.

[7] W. Charoenwet, P. Thongtanunam, V.-T. Pham, and C. Treude, "An Empirical Study of Static Analysis Tools for Secure Code Review," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Vienna Austria: ACM, Sep. 2024, pp. 691–703.

[8] M. Shen, A. A. Pillai, B. A. Yuan, J. C. Davis, and A. Machiry, "Finding 709 Defects in 258 Projects: An Experience Report on Applying CodeQL to Open-Source Embedded Software (Experience Paper) – Extended Report," Apr. 2025.

[9] G. Cardoen, T. Mens, and A. Decan, "A dataset of github actions workflow histories," in *Proceedings of the 21st International Conference on Mining Software Repositories*, ser. MSR '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 677–681. [Online]. Available: https://doi.org/10.1145/3643991.3644867

An In-Depth Analysis of the CodeQL Ecosystem

Jean-Charles Noirot Ferrand, Patrick McDaniel
University of Wisconsin–Madison

Semiconductor Research Corporation

MADS&P

Overview

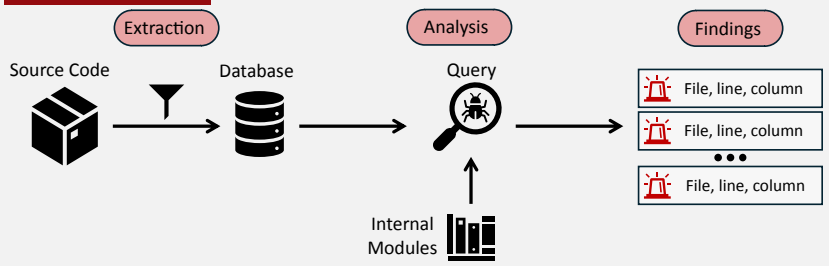
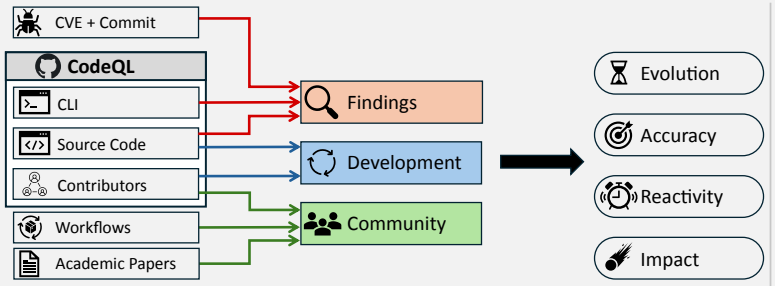


Figure 1: A CodeQL Analysis

- Software relies on **automated vulnerability detection** to improve the security
- **CodeQL** (from GitHub) is the most used static analysis tool
- We measure the full **ecosystem's health** through **three interconnected axes**

Methods



Data Sources

- **CodeQL**: Git and GitHub repository data, changelogs, CLI binaries
- **CVEFixes**: CVEs and corresponding fixing commit
- **Papers**: 1K+ papers mentioning "CodeQL"
- **Users**: 3M+ GitHub workflows histories

Three Axes

Security Rules

- Properties of **security rules**
  - **Accuracy:** Can the rule identify CVEs?
  - **Sophistication:** How many dependencies?
  - **Lifecycle:** How does a rule evolve?

Community

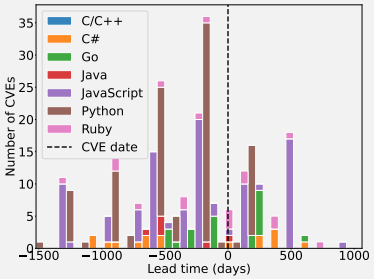
- Impact on academia and open-source software.
  - **Adoption:** Usage as the de facto static analyzer?
  - **In Practice:** How does it shape OSS/Academia?

</> Development

- Characterization of the evolution of the tool
  - **Developers:** Who contributes to the tool?
  - **Implications:** Practical impact on detection?

Measurement

Rules & Properties

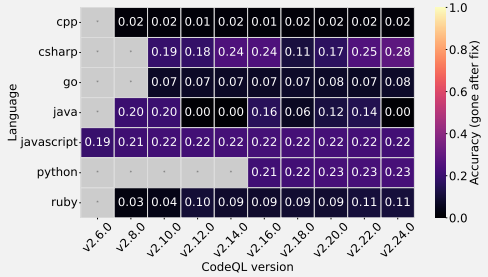


Impact on CVE detection

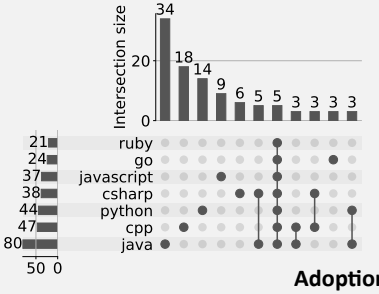
- Lead time to detect CVEs
- Mostly JavaScript/Python
- >50 CVEs could have been found
- **Limitation:** Automated filtering (same files and lines as fix commit)

Accuracy (CVEFixes)

- Low accuracy (comparable to prior studies), **little changes** across versions
- The **evolution** of OSS (languages, frameworks) needs to be reflected



Community & Development

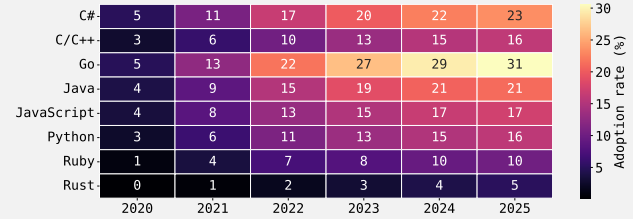


Language Expertise

- Heavy Java/C++ expertise
- Minority of one-language only contributors (42% for Java)
- **5** maintainers on all languages

Adoption Rate

- Study from GitHub Actions workflows histories of 40k+ repositories
- Observed **misalignment** between development and adoption (Go, C/C++)
- Start of adoption in 2022, convergence between 2024-2025



Future Work

Analysis Improvements

- Adoption from **papers** (academia) and **open-source software**
- Characterization of the number of findings (**sensitivity**)

Understanding Rules Development

- Characterization of a security rule and its scaffolding **evolution** across time → Addition/Deletion, Scope Improvements,...
- Improvement of internal library detection (**reachability**)

Security Implication

- Mapping from CodeQL changes to **CVE lead time**
- What matters the most when building **security heuristics** for open-source software program analysis?

Contact

https://jcnf.me
jcnf0
jcnf@cs.wisc.edu