LLM-Augmented Static Analysis Security Testing

Murilo Escher Pagotto Ronchi

Seminar: Software Quality
Advisor: Kohei Dozono
Technical University of Munich
murilo.escher@tum.de

Abstract. The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

Large language models (LLMs) have recently gained a lot of popularity, be it in academia, business or for personal use. Among their many use cases, one currently much researched possibility is their application in static application security testing (SAST).

Static application security testing is a widespread methodology for identifying software vulnerabilities by analyzing the source code, bytecode or binaries of an application. In contrast to dynamic testing methods, which require the code to be executed, SAST operates statically. This characteristic proves useful for integration into the software development lifecycle, for example in continuous integration (CI) pipelines. SAST tools, however, are often not capable of identifying all existing vulnerabilities and usually present many false negatives (FNs) [6] as well.

In light of this, the possibility of combining SAST tools' output with LLMs, as a way to enhance their reasoning and mitigate some of these flaws, is being much researched. These studies have investigated their use for taint analysis [5], combined SAST output with a retrieval augmented generation (RAG) system [1, 3], and compared the effectiveness of both tools [9].

This paper further investigates their usability when combined with LLMs. By leveraging a human-curated dataset containing known vulnerabilities in open-source software (OSS) [4], SAST tools [2, 7], and code context acquired through different program representations, this study evaluates the extent to which LLMs can enhance the accuracy and utility of static analysis results.

2 Related Work

Here it should be made clear how this project differs from related work. The ways in which we improved or completed other research should be made clearer.

Talk about repo-level being more important, [8]

3 Approach

How the tools were chosen, flow of information, etc

4 Evaluation

How the results are going to be interpreted. For ex. the voting system when using multiple SAST Tools. The research questions we are going to analyse also come here.

5 Conclusion

You can also reference other parts of the document, e.g., sections or subsections. In Section 1 we briefly introduced something, whereas in Subsection **??**, we motivated something else.

Make sure to capitalize chapters, sections or subsections when referencing them.

A Appendix

Anything additional goes here ...

Maybe extra information as to how the code words could be provided, if needed.

References

1. Du, X., Zheng, G., Wang, K., Feng, J., Deng, W., Liu, M., Chen, B., Peng, X., Ma, T., Lou, Y.: Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag (2024), https://arxiv.org/abs/2406.11147
2. GitHub: Codeql. https://codeql.github.com/, accessed: 2025-01-25
3. Keltek, M., Hu, R., Sani, M.F., Li, Z.: Boosting cybersecurity vulnerability scanning based on llm-supported static application security testing (2024), https://arxiv.org/abs/2409.15735
4. Kim, J., Hong, S.: Poster: BugOSS: A regression bug benchmark for evaluating fuzzing techniques. In: IEEE International Conference on Software Testing, Verification, and Validation (ICST) (2023)
5. Li, Z., Dutta, S., Naik, M.: Llm-assisted static analysis for detecting security vulnerabilities (2024), https://arxiv.org/abs/2405.17238
6. Lipp, S., Banescu, S., Pretschner, A.: An empirical study on the effectiveness of static c code analyzers for vulnerability detection. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 544–555. ISSTA 2022, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3533767.3534380, https://doi.org/10.1145/3533767.3534380
7. Meta: Infer: A tool to detect bugs in java and c/c++/objective-c code. https://fbinfer.com/, accessed: 2025-01-25
8. Risse, N., Böhme, M.: Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection (2024), https://arxiv.org/abs/2408.12986
9. Zhou, X., Tran, D.M., Le-Cong, T., Zhang, T., Irsan, I.C., Sumarlin, J., Le, B., Lo, D.: Comparison of static application security testing tools and large language models for repo-level vulnerability detection (2024), https://arxiv.org/abs/2407.16235