Maastricht University

Department of Advanced Computing Sciences

Adding Context to Automated Vulnerability Detection for Teaching Software Security

Antoine Dorard[A], Bastian Küppers[A], Ashish Sai[A], Theodor Schnitzler[A]

[A] Department of Advanced Computing Sciences

Background

Considering the recent developments in the field of generative AI, large language models (LLMs) can be leveraged to enhance static application security testing (SAST) tools and teaching about this topic. These models provide contextual information about identified vulnerabilities which can help students to differentiate genuine issues from false alarms while learning about software security. The approach includes analyzing vulnerabilities in android applications using SAST tools, clustering related code functionalities, and generating multi-level summaries for detected vulnerabilities. The process employs advanced clustering techniques and consensus-building methods to ensure accuracy.

Introduction

The security of programs is a crucial aspect in the development life-cycle of software applications. However, it seems that awareness of software vulnerabilities among students needs to be improved as they are apparently not aware of this kind of threat [1]. Therefore, students need to be educated about software vulnerabilities and especially need to learn two things: How to find security vulnerabilities and how to assess and fix them correctly. To assist in these steps, tools for Static Application Security Testing (SAST) are widely available [2]. SAST is a method for checking the security of an application, which scans the source code of a program for known vulnerable code patterns. However, SAST tools are known to have several drawbacks [3]. They especially tend to produce a high rate of false alarms [4, 5], which can be caused by a lack of analysis of the purpose of the application. This turns out to be a problem when integrating real-life examples of SAST tools into teaching, because real vulnerabilities are not easily distinguishable from false positives. Only by investing work in a manual inspection of the source code can real vulnerabilities be determined. While this may work for a prepared lecture, this yields a problem for hands-on tutorials and practical projects, leading to our research question:

How can context information be attributed to found security vulnerabilities?

Methods

The increase in performance in LLMs in recent years provides new ways for language and code generation, but also for understanding the, as modern LLMs can capture syntactic and semantic nuances in code far better than their predecessors. Therefore, we used LLMs, such as GTP-4o and Claude, in combination with the SAST tool Mobile Security Framework (MobSF). The resulting software pipeline includes a vulnerability analysis by MobSF, followed by a parsing step that prepares for a clustering. The purpose of clustering at this point is to group different program functionalities together and then associate each detected vulnerability with one of the identified clusters.

To adjust for problems that can arise during clustering, several clustering approaches are implemented, e.g. ACER Louvain Clustering [6] and their results are checked against each other to reach a consensus regarding existing clusters [7]. Once the clusters are generated, a summary is generated for each cluster containing at least one vulnerability, along with summaries for the classes and methods contained in the cluster. The summarization process takes all the methods and classes in a cluster and generates a summary describing the purpose of that particular part of the code. In total, three one-sentence summaries are generated, each at a different level of granularity. The general workflow is depicted in Figure 1.

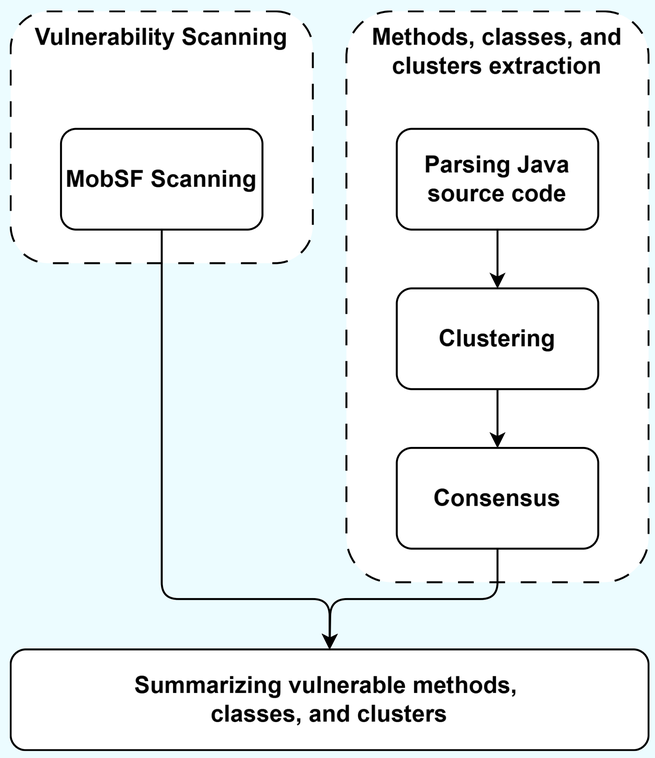

Figure 1: Overall Workflow

Conclusion

The integration of LLMs with SAST tools, in this case MobSF, has shown promise in reducing false positives and enhancing the learning process for software engineering and computer security students. The initial results suggest that providing contextual information significantly aids students in comprehending vulnerabilities. However, the small-scale study lacks statistical robustness, necessitating larger and more inclusive studies. Future work involves scaling the research, refining the framework, and assessing its real-world applicability to maximize its educational and practical benefits.

References

[1] Andrew Sanders, Gursimran Singh Walia, and Andrew Allen. 2024. Assessing common software vulnerabilities in undergraduate Computer Science assignments. DOI: 10.53735/cisse.v11i1.179

[2] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. 1999. Principles of Program Analysis. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-03811-6

[3] Kaixuan Li et al. 2023. Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java. DOI: 10.1145/3611643.3616262

[4] Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. DOI: 10.1145/2970276.2970347

[5] Brittany Johnson et al. 2013. Why don’t software developers use static analysis tools to find bugs? DOI: 10.1109/icse.2013.6606613

[6] Andrew Chen, Yanfu Yan, and Denys Poshyvanyk. 2023. ACER: An ASTbased Call Graph Generator Framework. DOI: 10.1109/SCAM59687.2023.00035

[7] Nam Nguyen and Rich Caruana. 2007. Consensus Clusterings. DOI: 10.1109/ICDM.2007.73

Correspondence to:
Dr. Bastian Küppers

b.kuppers@maastrichtuniversity.nl

Dept of Advanced Computing Sciences

T +31 43 388 34 94

Maastricht University

P.O. Box 616
6200 MD Maastricht, The Netherlands