Small Language Model Resilience to AST-Based Obfuscation Attacks on Reentrancy Vulnerability Detection

George Chidera Akor [1], Love Allen Chijioke Ahakonye [2], Jae Min Lee [1], Dong-Seong Kim [1] *
[1] IT-Convergence Engineering, *Kumoh National Institute of Technology*, Gumi, South Korea
[2] ICT Convergence Research Center, *Kumoh National Institute of Technology*, Gumi, South Korea
* NSLab Co. Ltd., Gumi, South Korea, *Kumoh National Institute of Technology*, Gumi, South Korea
(georgeakor, loveahakonye, ljmpaul, dskim@kumoh.ac.kr)

***Abstract*—The rise of Small Language Models (SLMs) presents opportunities for enhancing code security analysis, yet their reliability against adversarial attacks like code obfuscation remains critical [1]. This paper investigates the impact of semantics-preserving Abstract Syntax Tree (AST)-based identifier renaming on the reentrancy detection capabilities of two prominent 7B parameter SLMs: `codellama:7b-instruct` and `mistral:7b-instruct` [1]. Evaluating on a curated dataset of 50 Solidity contracts, we find contrasting results: `mistral:7b-instruct` exhibited high baseline performance ($F_1 \approx 0.93$) and remarkable robustness, with minimal performance degradation after obfuscation ($\Delta F_1 \approx -0.02$) [1]. Conversely, `codellama:7b-instruct` struggled at baseline ($F_1 \approx 0.39$) and displayed an anomalous performance *increase* post-obfuscation ($\Delta F_1 \approx +0.17$). Our contribution lies in providing direct quantitative evidence of significant variance in SLM robustness against a realistic obfuscation attack vector for vulnerability detection, highlighting the necessity of adversarial testing and motivating further research [1].**

***Index Terms*—Abstract Syntax Tree, Adversarial, Code Obfuscation, Detection, Reentrancy, SLM, Smart Contracts, Vulnerability**

I. Introduction

Small Language Models (SLMs) offer efficient solutions for sophisticated code analysis, with compelling potential for enhancing software security [1], especially in high-stakes domains like smart contracts where vulnerabilities such as reentrancy have caused massive financial losses [2]. However, the practical reliability of SLMs faces a critical challenge from adversarial code obfuscation techniques designed to evade detection [3], [4]. This paper focuses on a potent, semantics-preserving method: Abstract Syntax Tree (AST)-based identifier renaming, which replaces meaningful names with generic tokens (e.g., '_obf1') to defeat lexical analysis while keeping logic intact [5]. We investigate the resilience of two prominent 7B parameter SLMs, `codellama:7b-instruct` and `mistral:7b-instruct`, when faced with this obfuscation for the critical task of reentrancy detection in Solidity. To our knowledge, this is the first direct comparison evaluating these specific SLMs against this attack vector for this task. Our contributions are:

1) Quantifying the obfuscation's performance impact,
2) Revealing significant robustness differences between comparable SLMs, and
3) Uncovering an unexpected performance anomaly.

II. Methodology

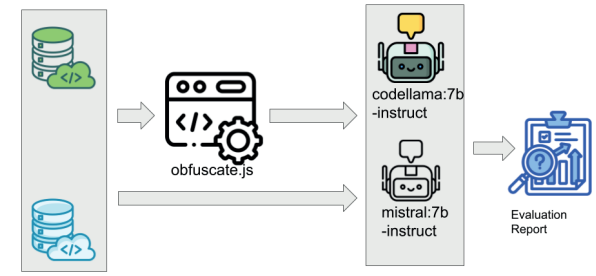

Fig. 1. Overview of concept experimentation

We evaluated two 7B parameter, instruction-tuned SLMs: M_1 = `codellama:7b-instruct` [6], [7] and M_2 = `mistral:7b-instruct` [8], [9], as shown in figure 1, executed locally via Ollama. The dataset D_{clean} comprised $N = 50$ curated Solidity contracts (from SmartBugs [10]), balanced with 25 reentrant ($y_c = True$) and 25 non-reentrant ($y_c = False$) examples. An obfuscated version D_{obf} was generated via AST-based identifier renaming using a Node.js script and '@solidity-parser/parser' , replacing all user-defined identifiers (contracts, functions, variables, etc.) with generic '_obfX' tokens while preserving code structure. Models performed binary classification ($f_{M_i} \quad D \rightarrow \{Yes, No\}$) based on a zero-shot, persona structured prompt.

Model performance was evaluated using the metrics below after mapping $Yes \rightarrow True, No \rightarrow False$. Responses parsed as 'Unclear' were excluded.

1) Precision ($P = \frac{TP}{TP+FP}$),
2) Recall ($R = \frac{TP}{TP+FN}$),
3) F1-Score ($F_1 = 2 \cdot \frac{P \cdot R}{P+R}$) , and
4) Accuracy ($Acc = \frac{TP+TN}{TP+FP+TN+FN}$)

calculated for the positive (reentrant) class.

III. Results and Discussion

The experiments revealed starkly different performance and robustness between the two SLMs, as shown in Fig. 2 and Fig. 3. On the original dataset (D_{clean}), M_2 (`mistral:7b-instruct`) demonstrated high effectiveness ($F_1 \approx 0.3$, $R \approx 0.7$), significantly outperforming M_1 (`codellama:7b-instruct`), which struggled ($F_1 \approx 0.3$, $R \approx 0.33$, $Acc \approx 0.47$).

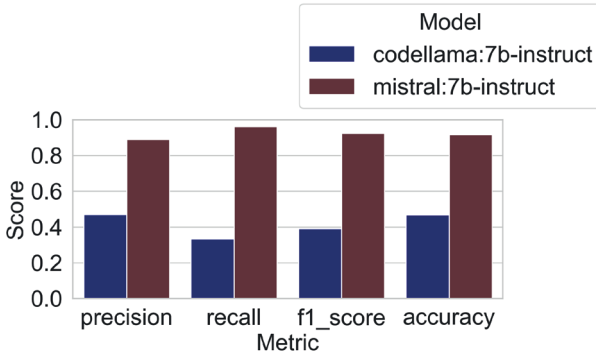

Fig. 2. Performance metrics on the original dataset D_{clean}.

Evaluating the impact of AST identifier renaming ($\Delta Metric = Metric_{obf} - Metric_{base}$) showed M_2 to be highly resilient. Its performance metrics barely changed on the obfuscated dataset D_{obf} ($\Delta F_1 \approx -0.02$), suggesting its detection relies on structural or control-flow patterns unaffected by name removal. Conversely, M_1 displayed an anomalous *improvement* post-obfuscation ($\Delta F_1 \approx +0.17$, $\Delta R \approx +0.17$), with an increase in 'Unclear' responses (from 3 to 8). While still underperforming M_2, this counterintuitive result might suggest the obfuscation removed initial misleading lexical cues or could be noise related to the small dataset (N=50).

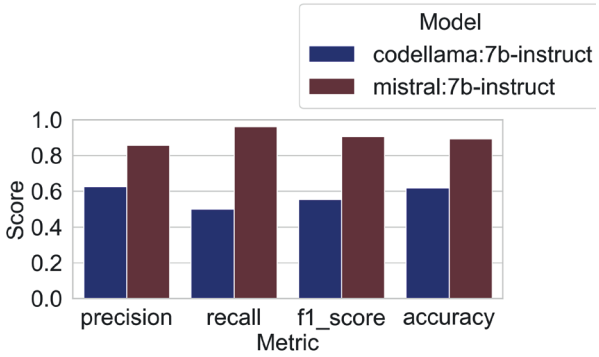

Fig. 3. Performance metrics on the AST-obfuscated dataset D_{obf}.

While this preliminary study is limited (N=50, one technique, two models), the potential for robust SLMs like Mistral-7B-Instruct warrants further investigation with larger datasets, diverse obfuscations, and more models, which we plan for future work.

IV. Conclusion

This preliminary study evaluated the robustness of `codellama:7b-instruct` and `mistral:7b-instruct` against AST-based identifier renaming for smart contract reentrancy detection. We found `mistral:7b-instruct` demonstrated strong baseline performance and high resilience. Conversely, `codellama:7b-instruct` exhibited poor baseline performance and anomalous improvement post-obfuscation. Our work provides direct quantitative evidence of variance in SLM robustness against this attack vector, and these results motivate future work involving larger datasets, diverse obfuscation techniques, and more models.

Acknowledgment

This work was partly supported by Innovative Human Resource Development for Local Intellectualization program through the IITP grant funded by the Korea government(MSIT) (IITP-2025-RS-2020-II201612, 33%) and by Priority Research Centers Program through the NRF funded by the MEST(2018R1A6A1A03024003, 33%) and by the MSIT, Korea, under the ITRC support program(IITP-2025-RS-2024-00438430, 34%).

References

[1] Anonymous, "A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness," *arXiv preprint arXiv:2411.03350*, 2024. [Online]. Available: https://arxiv.org/abs/2411.03350

[2] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Principles of Security and Trust (POST 2017)*, ser. ETAPS 2017. Springer, 2017, pp. 164–186.

[3] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," *Technical Report 148, Department of Computer Science, University of Auckland*, 1997. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.1300

[4] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pp. 297–300, 2010.

[5] A. Jha and C. K. Reddy, "Codeattack: Code-based adversarial attacks for pre-trained programming language models," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

[6] Meta AI, "Introducing codellama: A large language model for coding," https://ai.meta.com/blog/code-llama-large-language-model-coding/, 2023, meta AI blog post.

[7] Meta Llama Team, "meta-llama/codellama: Inference code for codellama models," https://github.com/meta-llama/codellama, 2023, gitHub repository.

[8] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[9] Mistral AI Team, "Announcing mistral-7b instruct: Instruction-tuned open-source llm," https://mistral.ai/news/announcing-mistral-7b, 2023, mistral AI blog post.

[10] M. di Angelo, T. Durieux, J. F. Ferreira, and G. Salzer, "SmartBugs 2.0: An execution framework for weakness detection in ethereum smart contracts," in *38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2023, pp. 2102–2105.