BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

Impact of AI Tools on Code Quality and Security

Peter Vinarčík*

Abstract
This work is aimed at creating a research framework focused on testing available generative AI. The output of the presented framework is a comprehensive overview of the tested AIs (ChatGPT-4, GitHub Copilot, ChatGPT-3.5, Gemini), mainly directed towards the security of the code generated by the given AIs. Another output is the correctness of the generated codes. Apart from the AI, framework integrates Static Application Security Testing (SAST) tools for finding vulnerabilities in the given code using static analysis. MITRE's methodology is then utilized to rate the severity of the vulnerabilities in the given code, in case any has been detected. The work also presents a second approach that targets on daily use of chat-bots such as ChatGPT or Gemini, where the integrated chat-bots are enhanced with an automatic vulnerability scan in the generated code.

*xvinar00@vut.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Only a small amount of researches targeting generative AI is focused on cybersecurity. Of these, most are performed on a single AI while testing is semi-automated or even manual. Therefore, there is a need to perform research on a larger scale and present a unified framework, usable for future testing extended with new AI or SAST tools.

2. Problem Definition

New models of generative AI are emerging every day, and with them comes the challenge of the security of the generated code. Existing researches described in chapter 3, do not agree on one particular approach for evaluating the safety and so this thesis aims to present a unified system for evaluating the safety of AI generated code.

3. Related Work

In the papers such as Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants [1], Do Users Write More Insecure Code with AI Assistants? [2], and Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions [3], all agree on the fact, that they were performed only with the use of a single AI - GitHub Copilot. In those cases, the evaluations are semi-automated or completely manual - with use of a security tool such as CodeQL. The above-mentioned studies have been an initial approach to the security, but the field of generative AI has experienced an enormous growth in the recent year, accompanied by the emergence of new models. The resulting framework is in some ways inspired by these studies, especially the tools used in these studies, but in addition to the large-scale experiment execution, the methodology for the evaluation of the generated code has been modified, which will be discussed and described further in section 4.

4. Proposed Solutions

This thesis presents two approaches.

4.1 Research Framework

The first is a research approach where a fully automated framework is introduced. The pipeline of the framework can be seen in **Figure 1**. The input is a dataset of human-readable prompts, or optionally differently formatted prompts separated by the delimiter, then the prompts are sent to the integrated AI modules. Subsequently, evaluation of the first statistic in the form of validity of the generated code takes place. An extension in this section is planned to introduce the possibility of using unit tests. In case the code is valid, a static analysis is performed. To support the vulnerability search, three SAST tools are integrated. Since more SAST tools are used and the static analysis is known to have a higher rate of false positives, the resulting analysis includes the intersection of the CWEs (Common Weakness Enumeration) found by the given tools to confirm if a certain vulnerability has been found by 2 or more tools. If the static analysis results with a vulnerability, this vulnerability is flagged with the appropriate Common Weakness Enumeration (CWE) by the SAST tool that discovered it. Using the NVD (National Vulnerability) API, all vulnerabilities linked to this CWE can be retrieved. These vulnerabilities are scored by the CVSS (Common Vulnerability Scoring System) and this system is being used by the implemented MITRE's methodology. The calculation of the severity of a given vulnerability can be observed in **Equation 1**. Using this methodology, vulnerabilities found with low frequency and high severity are rated low, because if developers are not making a particular mistake, then the vulnerability should not be rated high and vice versa.

4.2 Enhanced Chat-bots

The second approach is to integrate existing Chat-bots (currently Gemini, ChatGPT-3.5 and ChatGPT-4), where the responses that contain the source code are immediately scanned and the output is enhanced by marking the location of the potential vulnerabilities where they were found, with more details - see **Figure 2**.

5. Testing

Testing was performed on the dataset used in research [3], which contains a total of 29 Python and 25 C test cases, inspired by MITRE's Top 25 for 2021 (very few changes from 2023). The test cases consist of existing code that the AI is supposed to complete. The next iteration of testing is a planned on a dataset consisting from human-readable prompts, inspired by this dataset. The detailed results of each AI for each prompts are recorded with the schema as shown in **Figure 3**.

Proof-of-concept testing has currently been conducted, with a total of 4 different approaches. In Figure **Figure 4**, is the result of testing on the Python part of the dataset. The graph and the results marked with A are from the original dataset, which was taken from [3]. The graph and results B are experiments with a simple prompt-engineering approach, where a sentence is added to each prompt, pointing out the importance of code security. The sentence is formulated in general terms without a concrete warning about a specific vulnerability, since it is expected that in the real world, programmers using AI may not have a deeper knowledge about potential vulnerabilities.

In **Figure 5**, is testing on a C language dataset. The low percentage of vulnerable code is mainly explained by the fact that many of the prompts have not already passed the validity check of the generated code, and so the security analysis has not even taken place. However, the low percentage success rate does not necessarily imply the inability of the AI to generate C code, when a manual validation of the generated code was performed, it was observed that for C code, the AI often just completes a part of the required code (the used dataset is built on this principle as well) and so the compiler does not consider this code as valid. It was this testing result that initiated the motivation to prepare a dataset that will contain only sentences describing the expected result. An assumption for further research is that if the AI starts generating from scratch, the overall validity of the code will be higher.

6. Contribution

This thesis results in a proof-of-concept for testing generative AIs using the presented framework, which utilizes a MITRE-inspired methodology. It is thanks to this methodology that, compared to existing research, it is possible not only to determine the percentage statistics of secure and valid AI-generated code, but also to determine the severity of the reported vulnerabilities.

Moreover, an application which integrates existing chat-bots has been presented, where the user is alerted to a potential vulnerability during the generation of code by a given AI.

References

[1] Gustavo Sandoval, Hammond Pearce, Teo Nys, et al. Lost at c: A user study on the security implications of large language model code assistants. online, 8 2022.

[2] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with ai assistants? online, 11 2022.

[3] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, et al. Asleep at the keyboard? assessing the security of github copilot's code contributions. online, 8 2021.