Introduction:

A vulnerability in software is one that is a weakness in the system that could be exploited by malicious users. As technology advances, the threat for exploitations of systems increases as well. Vulnerabilities and exploitations are a serious challenge that software developers and organizations face. Generative AI, such as ChatGPT, are tools that can be used for vulnerability detection. There is the uncertainty for the accuracy and its ability to detect vulnerabilities. We will study the effectiveness of ChatGPT in detecting vulnerabilities in programs of codes. By studying the performance of GPT-4o-mini, we can better understand if it can be reliable for vulnerability detection.

There have been published research works[1] that are researched Generative AI for vulnerability detection. From what I've read, there has already been research in ChatGPT in predicting vulnerability compared to other Large Language Models(LLM). Early studies compared ChatGPT to other Large Language Models (LLMs) when such models were relatively new. While those studies highlighted the potential of ChatGPT, they were limited, the versions of ChatGPT used in earlier research differ vastly from the current models due to the exponential growth in LLM capabilities over recent years. That limitation, combined with the rapid evolution of LLMs, motivated us to investigate the relationship between code complexity, program size, and ChatGPT's accuracy in detecting vulnerabilities, offering a more comprehensive and up-to-date perspective. Since then, newer and more advanced versions of ChatGPT have been developed. From what I know, current research has not fully explored how factors such as code complexity and program size affect ChatGPT's ability to detect vulnerabilities accurately. In this paper, we will study the effect of code complexity, initially measured in terms of lines of code(LOC), on the accuracy of large language models like ChatGPT at detecting vulnerabilities. The approach that I used involves testing a dataset of C/C++ programs with known vulnerabilities using the GPT -4o-mini model and analyzing its ability to correctly identify these issues. In order to evaluate the model's performance, precision scores and trends in detection accuracy are looked at. By trying to correlate these factors, we aim to find patterns and limitations in its vulnerability detection capabilities. In the study, we will answer the following questions:

(RQ1) Is chatgpt able to detect vulnerabilities in the code?

Results: ChatGPT demonstrated the ability to detect vulnerabilities in the code.

(RQ2) Is the chatgpt reported vulnerability same as the recorded vulnerability in the dataset?

---

[1] Fu et al., "ChatGPT for Vulnerability Detection, Classification, and Repair"; Jensen, Tawosi, and Alamir, "Software Vulnerability and Functionality Assessment Using LLMs."

Results: With a precision score of 0.7551, ChatGPT correctly identified vulnerabilities in 75.51% of the analyzed files.

(RQ3) What is the relation between source code Lines of code (LOC) and chatgpt's ability to detect vulnerability correctly when comparing the recorded vulnerability?

Results: No significant relationship was observed between LOC and ChatGPT's accuracy in detecting vulnerabilities in the latest model, GPT-4o-mini.

(RQ4) What is the relation between source code complexity and chatgpt's ability to detect vulnerability correctly when comparing the recorded vulnerability?

Results: No relation between vulnerability detection ability and cyclomatic complexity. Contrary to belief, as complexity/LOC goes up, the detection ability gets better.

Related Work:

In order to understand the space of my research better, I have read numerous studies that have explored the use of Large Language Models (LLMs) for vulnerability detection and related software security tasks. In the paper, Software Vulnerability and Functionality Assessment Using LLMs[2], they evaluated nine LLMs on their ability to detect vulnerabilities and validate code functionality. The best proprietary model achieved 95.6% accuracy but had a low F1 score of 37.9%, indicating difficulty in describing vulnerabilities accurately. The main shortcoming was the focus on small code snippets, which limits applicability to real-world scenarios with larger programs.

ChatGPT for Vulnerability Detection, Classification, and Repair[3] paper studied ChatGPT's performance on various software vulnerability tasks and found it underperformed compared to specialized models, especially in generating repair patches. The lack of domain-specific fine-tuning and stochastic response variability were significant limitations, hindering its effectiveness for security tasks.

Prompting Large Language Models for Malicious Webpage Detection[4] paper explored LLMs like GPT-3.5 and ChatGPT for detecting malicious webpages through zero-shot and few-shot prompting. Few-shot prompting performed well, but ChatGPT sometimes failed to respond in zero-shot tasks. The study's reliance on a single dataset and limited scope to phishing and hacked webpages reduced the generalizability of its findings.

Software Vulnerability Detection Using Large Language Models[5] paper studies LLMs for detecting SQL injection and buffer overflow vulnerabilities, finding that fine-tuning improved

[2] Jensen, Tawosi, and Alamir, "Software Vulnerability and Functionality Assessment Using LLMs."
[3] Fu et al., "ChatGPT for Vulnerability Detection, Classification, and Repair."
[4] Li and Gong, "Prompting Large Language Models for Malicious Webpage Detection."
[5] Purba et al., "Software Vulnerability Detection Using Large Language Models."

performance but resulted in high false positive rates. The reliance on datasets with data quality issues, like superficial fixes, and limited focus on specific vulnerabilities were major shortcomings.

Approach:

This study investigates the ability of ChatGPT to detect vulnerabilities in C and C++ code while exploring the relationships between source code characteristics, such as lines of code (LOC) and complexity, and ChatGPT's detection performance. The research focuses on the following questions:

RQ1. Is ChatGPT able to detect vulnerabilities in the code?

RQ2. Does the vulnerability reported by ChatGPT match the recorded vulnerability in the dataset?

RQ3. What is the relationship between source code Lines of Code (LOC) and ChatGPT's ability to detect vulnerabilities correctly when compared to the recorded vulnerabilities?

RQ4. What is the relationship between source code complexity and ChatGPT's ability to detect vulnerabilities correctly when compared to the recorded vulnerabilities?

Dataset Selection Process

At the outset, we considered using the BigVul dataset due to its comprehensive links to GitHub commits and detailed metadata on vulnerabilities. However, challenges such as the dataset's large size and difficulties in interpreting columns like "function-before" and "function-after" led us to switch to a more manageable dataset. The chosen dataset, "A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries,[6]" provided a clearer structure and was better suited to our research. We specifically worked with the all_c_cpp_release2.0.csv file from this dataset, downloading 100 samples for analysis. This dataset offered detailed mappings of vulnerable code segments to their corresponding CVE and CWE classifications, ensuring relevance to our objectives.

Data Collection and Preprocessing

To automate the dataset collection and analysis, we utilized Jupyter Notebook as the primary environment. Automation scripts were developed to streamline the downloading of code files based on commit IDs provided in the dataset. The scripts dynamically adjusted API wait times to

manage rate limits effectively. Verification steps ensured that the downloaded files corresponded to their intended vulnerable versions by cross-referencing commit dates with dataset metadata.

Analysis Process

The OpenAI API played a central role in testing the code snippets against ChatGPT. Each snippet was evaluated to determine whether ChatGPT could correctly identify vulnerabilities. The Lizard library was employed to measure code complexity metrics such as cyclomatic complexity and LOC, providing additional insights into the relationship between these factors and ChatGPT's detection performance.

Files were categorized into complexity ranges based on LOC (e.g., 0-50, 50-100, and so on) and examined for detection performance. Detection performance was visualized using graphs where the X-axis represented complexity ranges, and the Y-axis indicated the detection rate (calculated as the number of vulnerabilities detected divided by the total number of files in that range). While no significant correlation was observed between complexity and detection performance, this analysis provided a nuanced understanding of ChatGPT's capabilities.

Quality Assurance and Documentation

Throughout the study, precision was calculated as the number of times ChatGPT's classification matched the recorded classification divided by the total number of files examined. Spot-checks confirmed that the correct versions of files were downloaded, ensuring the reliability of our dataset. Additionally, unnecessary files, such as changelog.txt, were removed from analysis spreadsheets for clarity, and README files were prepared to document the dataset and analysis processes.

Results:

RQ1: Is ChatGPT able to detect vulnerabilities in the code?

To answer this question, we ran 100 code snippets from the dataset through ChatGPT. The model successfully detected vulnerabilities in the majority of cases, demonstrating its capability to identify potential issues in C and C++ code. This was assessed by comparing ChatGPT's output to the dataset's recorded vulnerabilities, providing initial evidence of its effectiveness.

RQ2: Does the vulnerability reported by ChatGPT match the recorded vulnerability in the dataset?

We computed precision as the primary metric to evaluate the alignment between ChatGPT's reported vulnerabilities and those recorded in the dataset. Precision was calculated as the number

of correct detections divided by the total number of files analyzed. ChatGPT achieved a precision score of 0.7551, indicating that it correctly identified vulnerabilities in 75.51% of the analyzed files. This highlights reasonable alignment but suggests room for improvement.

RQ3: What is the relationship between source code Lines of Code (LOC) and ChatGPT's ability to detect vulnerabilities correctly?

Using graphs that plotted LOC ranges (X-axis) against detection rates (Y-axis), we examined whether larger or smaller programs influenced ChatGPT's detection accuracy. Contrary to expectations, no significant relationship was observed between LOC and detection performance. The analysis revealed that detection rates remained consistent across varying LOC ranges, suggesting that code size does not significantly impact ChatGPT's accuracy.

RQ4: What is the relationship between source code complexity and ChatGPT's ability to detect vulnerabilities correctly?

Cyclomatic complexity was used as a metric to evaluate code complexity. Similar to LOC, we graphed complexity ranges against detection rates. The results indicated no significant relationship between complexity and detection accuracy. Interestingly, there was a trend suggesting that as complexity increased, detection performance slightly improved, though this was not statistically significant. This challenges conventional beliefs that higher complexity negatively impacts vulnerability detection.

Threats to validity:

Several assumptions and limitations were inherent in our study. One key assumption was that the "function-before" column from the dataset accurately represented the vulnerable code segment. This assumption was necessary due to the lack of granular details in some dataset entries. Additionally, our reliance on a single dataset limited the generalizability of the findings. While the dataset provided valuable insights, its scope might not fully represent the diversity of real-world vulnerabilities.

Another potential threat to validity was the way ChatGPT's output was evaluated. We marked a detection as correct if ChatGPT identified at least one vulnerability matching the dataset classification. However, ChatGPT often reported additional vulnerabilities not classified in the dataset, which were neither validated nor accounted for in our precision metric. This might have led to an underestimation of ChatGPT's true capability to detect vulnerabilities.

The study was further limited by ChatGPT's token limit, restricting analysis to files with fewer than 3000 lines of code. This excluded larger programs that could have provided additional insights into the relationship between LOC and detection performance. Moreover, the absence of experiments with other LLMs, such as CodeBERT or AIBugHunter, limited comparative analysis, leaving potential gaps in understanding how ChatGPT performs relative to specialized models.

In future studies, expanding the dataset scope, validating additional vulnerabilities reported by ChatGPT, and exploring its performance on larger code files would enhance the robustness of the findings. Incorporating experiments with other LLMs would also provide a broader perspective on the effectiveness of generative AI in vulnerability detection.

Conclusion:

This study investigated ChatGPT's ability to detect vulnerabilities in C and C++ code, focusing on its performance in relation to code complexity and program size. While ChatGPT demonstrated a reasonable precision of 75.51%, no significant correlations were observed between its accuracy and factors like lines of code or cyclomatic complexity. Interestingly, the trend of improved performance with increasing complexity challenges conventional assumptions. These findings highlight both the potential and limitations of using ChatGPT for vulnerability detection, paving the way for further research into improving its reliability and extending its applicability to broader contexts.

To ensure transparency and reproducibility, all scripts, data preprocessing steps, and analysis tools used in this study will be made available on GitHub. This includes the automation scripts for dataset collection, code snippets analyzed, and any related documentation. The GitHub repository will also include this paper, serving as a comprehensive resource for researchers interested in exploring this area further.

References:

6. ZeoVan. *A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries*. GitHub, https://github.com/ZeoVan/MSR_20_Code_vulnerability_CSV_Dataset/blob/master/README.md.