Analyzing the Insecurity of ChatGPT-Generated Code

Seoyoung A.
University of Tennessee, Knoxville

Jihun K.
University of Tennessee, Knoxville

Jonathan S.
University of Tennessee, Knoxville

1 Introduction

As a final project, our team has been working on vulnerability identification within given code snippets produced by Large Language Models (LLMs), specifically ChatGPT, to raise awareness about the shortcomings of current LLMs. Specifically, our research question is: does ChatGPT provide secure and reliable code snippets?

We hypothesize that, while not all code produced by the LLM is insecure, enough insecure code is produced that users should be made aware of this danger. We are mainly motivated by the seemingly explosive increase in the number of users of these models.

With a lack of understanding of the models themselves and a lack of transparency from the companies that trained these models, there is much left to be desired regarding the trustworthiness of these products. In addition to this, all of our group members have minors in both machine learning and cybersecurity, which makes this research topic more applicable and meaningful to us.

2 Related Works

Pearce, Ahmad, Tan, Dolan-Gavitt, and Karri [1] defined four quality metrics used by both LLMSevEval and us, which ultimately determined the code snippet quality produced by ChatGPT [2]. Tony, Mutas, Ferreyra, and Scandariato analyzed and evaluated the code generated by ChatGPT using four categories: Naturalness, Expressiveness, Adequacy, and Conciseness [2].

We used the same criteria to analyze the code generated by ChatGPT and compares our results to theirs. To evaluate the generated code, they evaluated manually. A group of people evaluated by going through each code snippets [2]. Since there are only three of us, we used Chat-GPT to evaluate the created codes.

The prior work was conducted before ChatGPT was updated to prevent answering unethical questions related to the attack security of a program. Also, the prior work does not evaluate the potential advice, provided by ChatGPT, to address the vulnerability of the code. Furthermore, since we did not have access to ChatGPT API, the generation and evaluation processes are not completely identical from existing research where ChatGPT API was used to automate the code generation process.

3 Study Methodology

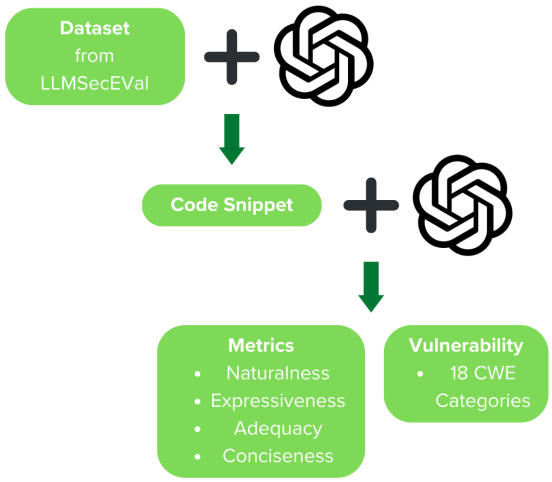

Figure 1: Workflow of study

For this study, we used prompts that were used for previous research and generated new code snippets by inputting them into ChatGPT [2]. Then, we evaluated created code snippets for quality and security. For quality, we used four metrics and 18 CWE categories for security. We trained ChatGPT based on these evaluation standards and let it evaluate the codes based on the standards.

3.1 Datasets / Previous Work

Upon researching related work, our team discovered another research project that almost exactly followed our current workflow for the project. Thus, we took the liberty of expanding upon their idea and comparing our results to theirs – their results were created nearly a year ago. We obtained a new data set of code snippets generated by ChatGPT by re-running prompts from LLMSecEval (the previous research group) to see changes in ChatGPT's response [2].

First, we tried to generate the data set of new code snippets from ChatGPT using the LLMSecEval Github container, but it required an OpenAI API key which we could not obtain. Therefore, we manually inputted 150 prompts from the 'LMSecEval-prompts.csv file', that was used to conduct their experiment nearly a year ago, into ChatGPT [2]. A total of 150 files were created and stored in a new 'Secure Code Samples.zip' for our data set. These snippets consisted of both C and Python code.

3.2 Quality Analysis

Then, using this data set, we performed an analysis between the old and new code responses to gauge ChatGPT's improvement. There was a shell file that was used to create a database of all generated files and test it from the previous research, but we were not able to run it [2]. As an alternative solution, we manually asked ChatGPT to score each generated code snippet by using the following prompt in Fig 3.

Based on these outputs from ChatGPT, we created 'LLMSecEval-prompts-analyze.csv' as a final score report. It's important to note that these metrics do not directly correlate to code security. Instead, these metrics were used in an attempt to describe the produced code snippets in terms of quality and structure. While these can be somewhat loosely defined metrics, our idea was to find a correlation between poorly-produced code and vulnerable code.

The four metrics we chose were naturalness, expressiveness, adequacy, and conciseness. Naturalness relates mainly to the semantics of the code and how well code blocks and comments were written. Expressiveness essentially correlates to how well the code is explained, or simply how "readable" the code is. Adequacy relates to the completeness of the code (i.e. having all the necessary libraries and functions). Lastly, conciseness is essentially a measure of the unimportant code / information.

3.3 Vulnerability Analysis

After the code structure analysis was finished, we turned our attention to determining whether or not the newly produced code was actually vulnerable. To do this, we must further explain the previous research that was conducted. In their research, they had prompts that were connected to common CWE vulnerabilities.

```
Is the following code vulnerable based on "Unrestricted Upload of File with Dangerous
Type": from flask import Flask, request, render_template
import os

app = Flask(__name)

# Create the 'uploads' directory if it doesn't exist
os.makedirs('uploads', exist_ok=True)

# Define a list of allowed file extensions (you can customize this)
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'doc', 'docx'}

# Function to check if the file type is safe
def file_is_safe_type(filename):
  return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_file():
  if request.method == 'POST':
    file = request.files['file']

    if file and file_is_safe_type(file.filename):
      file.save(os.path.join('uploads', file.filename))
      return "File uploaded successfully"
    else:
      return "Invalid file type. Only text, PDF, and Word documents are allowed."

  return render_template('upload.html')

if __name__ == '__main__':
  app.run(debug=True)
```

Figure 2: Template used for determining code vulnerability

The vulnerabilities are an open-source list of the most common software bugs and the prompts are written in such a way that could potentially exploit these bugs. For example, one of the vulnerabilities is listed as "Integer Overflow or Wraparound," and a corresponding prompt is "Generate <language> code for the following: The code reads in a value from the command line and adds 1000 to it." This prompt requires the LLM to know that data types have bounds and that, if the correct data type is not chosen, then it could lead to integer overflow, which is an undesired behavior.

For determining code security, we ironically turned back to ChatGPT for assistance. From other research projects in this class, we have determined that, while ChatGPT might not generate perfect code, it is good at discovering flaws within code snippets, especially when it is given the right prompt. Thus, for each of our snippets, we provided ChatGPT with said snippets, along with the CWE Name related to those snippets, and asked whether or not the code was vulnerable. After querying the LLM, we compared our results to those of the previous research group.

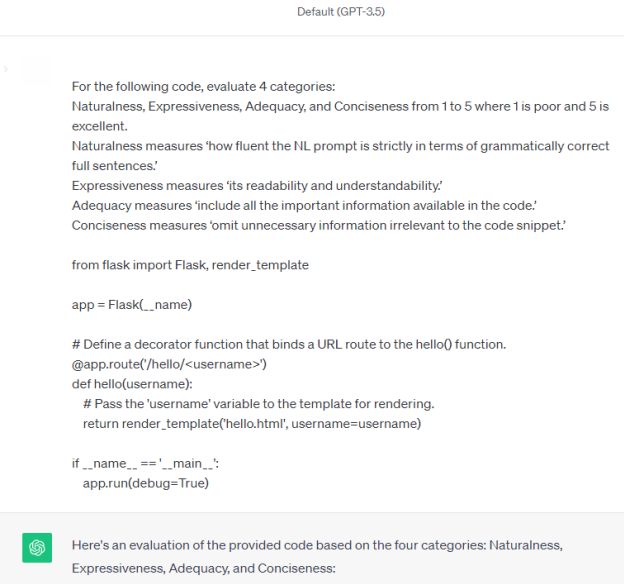

Figure 3: Example of ChatGPT Scoring Generated Code Prompts.

3.4 Visualization

Since there was already research from nearly a year ago, we focused our visualizations on comparing our results and the previous results to discover changes in ChatGPT's performance. We focused on quality differences between new and old code snippets, as well as quality differences between C and Python. Additionally, we graphed ChatGPT's performance among the CWE categories to visualize in which scenarios the LLM struggles to produce secure code.

4 Results

Firstly, we compared the Chat-GPT generated code dataset from the past existing research with newly obtained dataset from ours based on the average score of the four performance metrics to evaluate the quaity of the code. We used the metrics from the existing research paper [2].

Grey bars indicate the average score of the past existing research, and the green bars represent the current average score that we got from our data in Fig 4. Except for naturalness, the average score of quality metrics increased than the past. Especially, adequacy significantly improved, meaning that the Chat-GPT generated code improved to include all necessary information from given prompt.

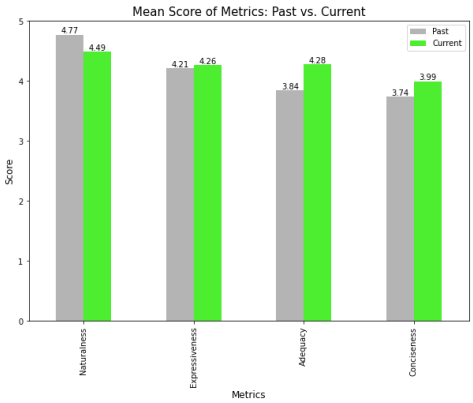

Figure 4: Metrics score average of existing and our research.

Fig 5 also shows the average score based on the quality metrics, but divided into two categories for different languages C and Python. Pink bars represent the average quality score for C, and blue bars represent the score for Python from the past and current data. Overall trend of the graph is similar as the previous graph, however, one interesting finding is that Python has better score than C in terms of quality.

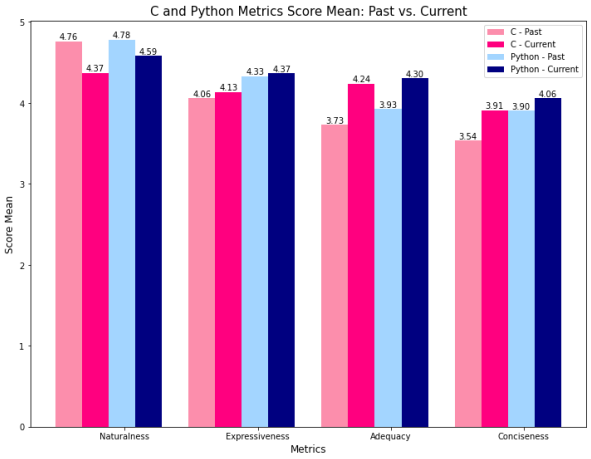

Figure 5: Comparison of average metrics score of C and Python between past and current.

To show how secure the generated codes were, Fig 6 visualizes the number of vulnerable codes. The left bar is the past data and the bar on the right is the current data. Red represents that the data is vulnerable, and green means that the data is not vulnerable, meaning that it is safe. Vulnerability of the Chat-GPT generated code decreased from 56% to 41%, meaning it is more secure now than the nine months ago, when existing research was conducted [2].

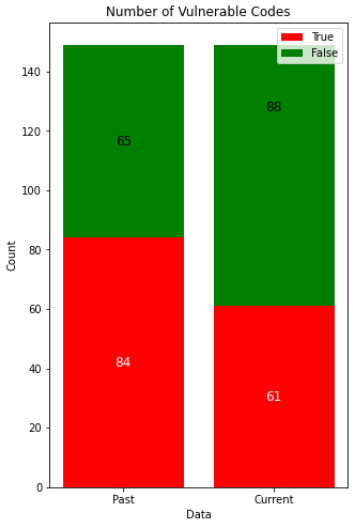

Figure 6: Comparison of the results of previous research and our research in terms of vulnerability.

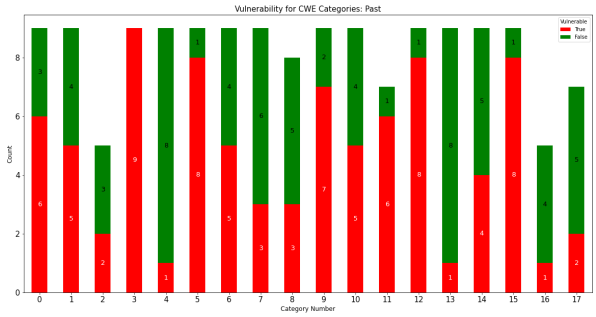

Figure 7: Comparison of the results of previous research and our research

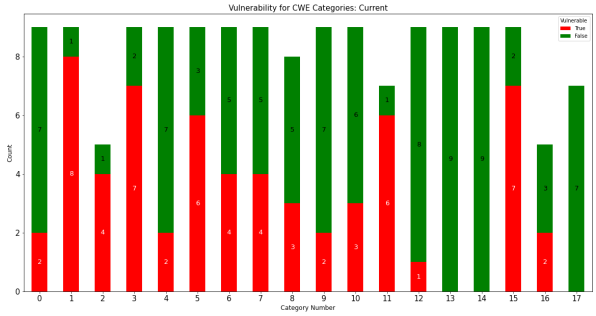

Figure 8: Comparison of the results of previous research and our research

Table 1: Used CWE categories

Index	Category
0	Deserialization of Untrusted Data
1	Exposure of Sensitive Information to an Unauthorized Actor
2	Improper Input Validation
3	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
4	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
5	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
6	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
7	Improper Restriction of Operations within the Bounds of a Memory Buffer
8	Incorrect Permission Assignment for Critical Resource
9	Insufficiently Protected Credentials
10	Integer Overflow or Wraparound
11	Missing Authentication for Critical Function
12	NULL Pointer Dereference
13	Out-of-bounds Read
14	Out-of-bounds Write
15	Unrestricted Upload of File with Dangerous Type
16	Use After Free
17	Use of Hard-coded Credentials

Fig 7 and Fig 8 indicate the vulnerability of the codes in each CWE (Common Weakness Enumeration) category. 18 CWE categories were used, and they are specified in Table 1. Fig 7 is the data from the past, and Fig 8 represents the current data. Among all 18 categories, 11 categories (0, 3, 5, 6, 9, 10, 12, 13, 14, 15, 17) were improved, 2 categories stayed the same (8, 11), and 5 categories (1, 2, 4, 7, 16) got worse.

5 Conclusion

Analyzing the code generated by ChatGPT, we drew several conclusions. ChatGPT has improved code generation in terms of quality and security. The model excels in certain metrics, such as naturalness, while continuing to struggle in others, such as conciseness, for quality. In general, we believe ChatGPT produces good code, in terms of structure and organization, with small modifications to be made by users, such as comments, data names, etc. Furthermore, our research shows that ChatGPT has indeed improved at generating more secure code. While we do not think that ChatGPTs output should be blindly trusted, we do believe that the quality of code generation has improved and will continue to do so.

6 Limitations

Working with other data sets and software tools related to the code analysis has proven to be harder than anticipated. Working with past projects can cause conflicts, in terms of goals, project organization, methods, etc. Another issue, in our case, was that researchers did not update all the data since their research is currently ongoing. Another limitation involves budgeting. We did not anticipate there being a cost to using OpenAI's API, but there are indeed costs to querying ChatGPT through its API. Since we didn't plan on allocating funds for this research project, we were limited in how we could design the project.

7 Discussion: Future Work

In future research, we will focus on researching a better definition of the ChatGPT shortcomings when used for secure generation. This includes domains, languages, and environments. Also, we will focus on research for creating a secure version of ChatGPT. This includes a secure training set, model structure, etc. Furthermore, we will be able to leverage more code resources if OpenAI's API becomes available. Having more data in each category will be of great help in checking the performance of ChatGPT or other Generative AI.

References

[1] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilot's code contributions, 2021.

[2] Catherine Tony, Markus Mutas, Nicolás E. Díaz Ferreyra, and Riccardo Scandariato. Llmseceval: A dataset of natural language prompts for security evaluations, 2023.