Step-by-Step Vulnerability Detection using Large Language Models

Saad Ullah [1] Ayse Coskun [1] Alessandro Morari [2] Saurabh Pujar [2] Gianluca Stringhini [1]

[1]Boston University [2]IBM Research

Motivation

- Vulnerability detection is a very critical task for systems security.
- Current analysis techniques suffer from the trade-off between coverage and accuracy.
- ML-based* analysis tools are non-robust, black-box and unreliable to use in real-world [1].
- LLMs* demonstrate revolutionizing capabilities for programming language-related tasks but they are also studied in a *black-box fashion* for both vulnerability detection and its repair.
- Security experts follow a step-by-step approach for vulnerability detection. Can using the same approach help LLMs performing better at the vulnerability detection task?

Objective

Design a framework to emulate step-by-step reasoning process of a human security expert using LLMs, to efficiently detect vulnerabilities in source code.

Methodology

- Our approach uses few-shot in-context learning to guide LLMs to follow a step-by-step human-like reasoning model for vulnerability detection.
- We make sure that the model first generates chain-of-thought reasoning [5] and then makes a decision based on that reasoning (as shown in Figure 1 and 3b).

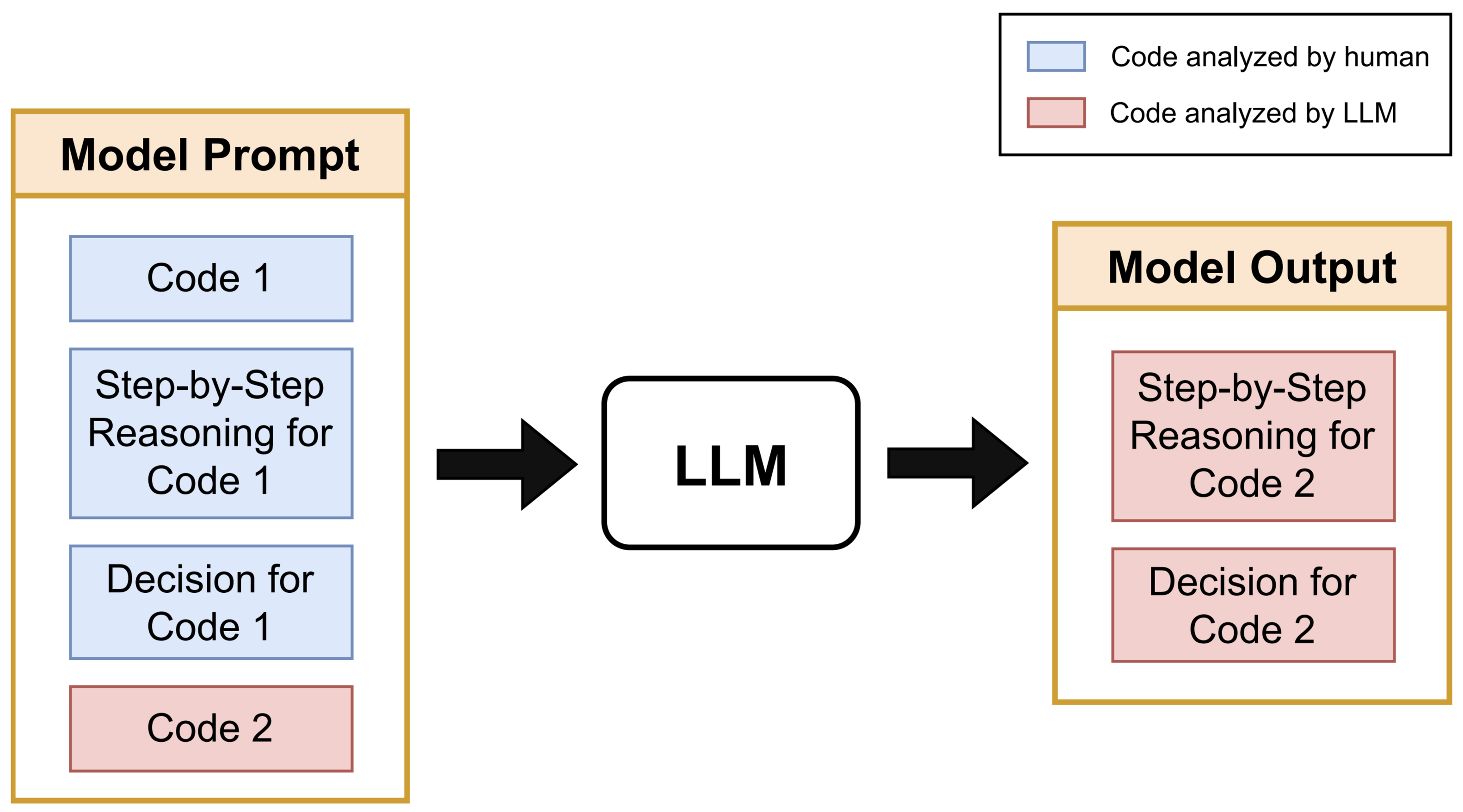

Figure 1. Overview of our few-shot in-context learning approach for vulnerability detection using LLMs.

* ML = Machine Learning
LLM = Large Language Model

Visualizing the Process of Vulnerability Detection

In this experiment, we study the behavior of an LLM when it is asked to detect a vulnerability in two different scenarios. First, when it is asked to give a direct answer (Figure 3a); and second, when it is first asked to perform human-expert like reasoning and then make a decision (Figure 3b).

We choose GPT-3.5 as an LLM and a code snippet (shown in Figure 2) containing an out-of-bound write vulnerability as a running example.

```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);

    /*routine that ensures user_supplied_addr
    is in the right format for conversion */

    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

Figure 2. Code Snippet from the MITRE Out-of-Bound Write Vulnerability (CWE-787).

Model Prompt

Q: Code (Figure 2): ...
Is the given code potentially vulnerable to out-of-bounds write?

Model Output

A: No, the given code is not potentially vulnerable to out-of-bounds write. ...

(a) Standard Prompting.

Model Prompt

Q: Vulnerable code snippet example: ...
A: Step-by-step analysis by human: ...
Decision by human on the presence of vulnerability: ...
Q: Code (Figure 2): ...

Model Output

A: Overview:

It is a function called `host_lookup` that performs a *reverse DNS lookup* to find the hostname associated with a given IP address.

Sub-components:

...
3. "*char hostname[64]*" is an array that will store the hostname.
4. "*inet_addr*" converts an IP address string to its binary form.
5. "*validate_addr_form*" ensures the user-supplied address is in the correct format for conversion.

Analysis:

1. ...
2. The code does not check if "*gethostbyaddr*" returns NULL. If it does, accessing "*hp->h_name*" will result in a segmentation fault.
3. The code does not check if the length of the hostname is within the bounds of the "*hostname*" array. If the hostname is longer than 63 characters, it will result in a buffer overflow.

Decision:

Based on the above analysis steps, the code is vulnerable to segmentation faults and buffer overflow ...

(b) Step-by-Step Reason Prompting.

Figure 3. Vulnerability Analysis using GPT-3.5

Evaluation

- Figure 3 shows that step-by-step reasoning guides the LLM to detect the (CWE-787) vulnerability.
- To systematically evaluate this approach, we create our own diverse synthetic dataset based on a subset of the MITRE 2022 top 25 most dangerous vulnerabilities.
- For each vulnerability we create vulnerable examples and their patches with varying levels of complexity.
- We use the 'gpt-3.5-turbo-16k' chat API to compare our approach with SoTA tools (Table 1).

Tool/Model	Description	Size	F1	Precision
cppcheck, infer, flawfinder	Combination of SoTA static analysis (SA) tools for C/C++	-	0.49	0.53
UniXcoder	RoBERTa-based model fine-tuned for defect detection in C/C++	126M	0.33	0.25
CodeT5+	LLM specifically pre-trained for progamming langauges-related tasks, including C/C++	16B	0.46	0.54
GPT-3.5	GPT-3.5 without reasoning	175B	0.48	0.50
Our approach with GPT-3.5	GPT-3.5 with step-by-step reasoning	175B	**0.70**	**0.72**

Table 1. Evaluation of different vulnerability analysis techniques on our dataset.

Takeaways

- Following a human-like step-by-step reasoning approach helps LLMs to efficiently analyze code and detect vulnerabilities.
- Our approach provides an explanation for the detected vulnerabilities, which helps user to better contextualize them and to find their root cause.
- Systematic evaluation of this approach on real-world datasets is still required to determine its reliability in real-world use cases.

References

[1] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, and Fabio Pierazzi. Dos and don'ts of machine learning in computer security, 2021.

[2] Mark Chen, Jerry Tworek, and Heewoo Jun. Evaluating large language models trained on code, 2021.

[3] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek. An observational investigation of reverse Engineers' processes, 2020.

[4] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. Hackers vs. testers: A comparison of software vulnerability discovery processes, 2018.

[5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[saadu, gian, acoskun]@bu.edu amorari@us.ibm.com saurabh.pujar@ibm.com