Reproducing CVEs using the power of LLMs

Course project for CS279

Praneeth, Laasya, Tim, Jiabao, Yujian

November 13, 2024

Problem Statement

- Software systems increasingly rely on crowd-sourced vulnerability reports(CVEs).
- Reproducibility of these vulnerabilities is often uncertain and challenging.
- Key Issues:
  - Incomplete information in vulnerability reports.
  - Manual effort required to reproduce vulnerabilities.

Why is it a Problem?

- Impact of Low Reproducibility:
  - Security analysts spent 3600 man-hours to reproduce 368 vulnerabilities.[1]
  - Only a fraction of vulnerabilities were successfully reproduced due to incomplete reports.
- Challenges Highlighted:
  - Manual debugging and speculative guessing are often necessary.
  - Lack of automated mechanisms to fill in missing information.
  - Hard to pinpoint the vulnerable function.

---

[1] Mu, Dongliang, et al. "Understanding the reproducibility of crowd-reported security vulnerabilities." 27th USENIX Security Symposium (USENIX Security 18). 2018.

What is Being Done Currently?

- Current Efforts:
  - Utilization of datasets like ARVO, which reproduces over 5,000 vulnerabilities across 250 projects.[2]
  - Implementation of reliable re-compilation systems to ensure vulnerabilities can be reproduced and tested.[3]
  - Crowdsourcing information gathering to improve reproduction success rates.
  - Localize vulnerable functions based on CVE descriptions and source code using LLMs.[4]

---

[2] Mei, Xiang, et al. "ARVO: Atlas of Reproducible Vulnerabilities for Open Source Software." arXiv preprint arXiv:2408.02153 (2024).

[3] CVEX

[4] Wu, Yulun, et al. "Effective Vulnerable Function Identification based on CVE Description Empowered by Large Language Models." 39th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2024.

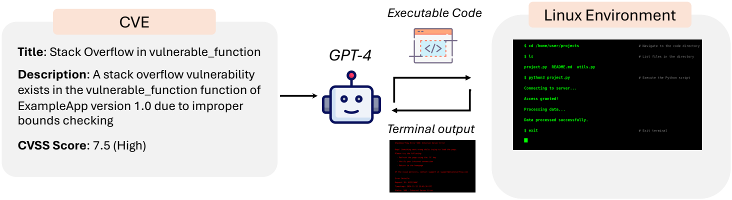
Why is it Not Enough?

- Limitations of Current Approaches:
    - Manual efforts are time-consuming and not scalable.
    - Despite crowdsourcing, key challenges remain in troubleshooting non-reproducible cases.
    - Existing datasets are often outdated.
- Need for Improvement:
    - Automated mechanisms are necessary to enhance the completeness and accuracy of vulnerability reports.
    - Current datasets, while valuable, need to be more comprehensive and easily updatable.

Our Solution

- Proposed Approach:
    - Utilize Large Language Models (LLMs) to automate the reproduction of vulnerabilities from their CVEs.

LLM Agents Experimenting with CVE



Title: Stack Overflow in vulnerable_function

Description: A stack overflow vulnerability exists in the vulnerable_function function of ExampleApp version 1.0 due to improper bounds checking

CVSS Score: 7.5 (High)

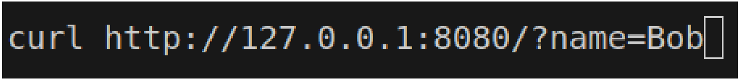
```
$ cd /home/user/projects                    # Navigate to the code directory
$ ls                                        # List files in the directory
project.py  README.md  utils.py
$ python3 project.py                        # Execute the Python script
Connecting to server...
Access granted!
Processing data...
Data processed successfully.
$ exit                                      # Exit terminal
█
```

Figure: Illustration of the LLM CVE exploit flow

Why is it Better?

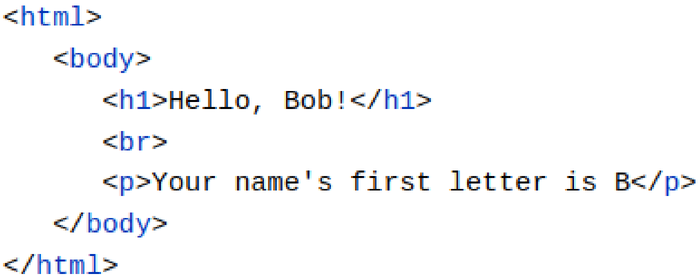
- Advantages of Our Solution:
  - Reduces manual effort and increases scalability.
  - LLMs can process and analyze large volumes of data quickly, improving accuracy.
- Comparison with Existing Solutions:
  - More efficient than manual debugging and speculative guessing.
  - Provides a dynamic and continuously updating dataset.

Prove it Experimentally



```
curl http://127.0.0.1:8080/?name=Bob
```

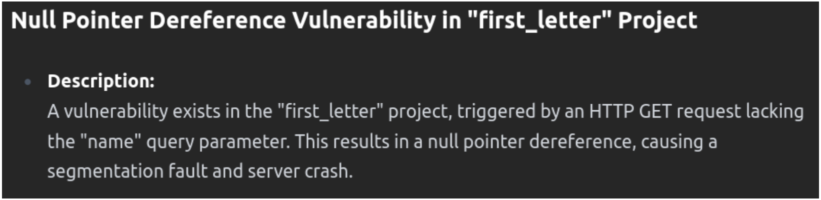
Figure: Vulnerable app for testing



```
<html>
   <body>
      <h1>Hello, Bob!</h1>
      <br>
      <p>Your name's first letter is B</p>
   </body>
</html>
```

Figure: Sample response

Prove it Experimentally

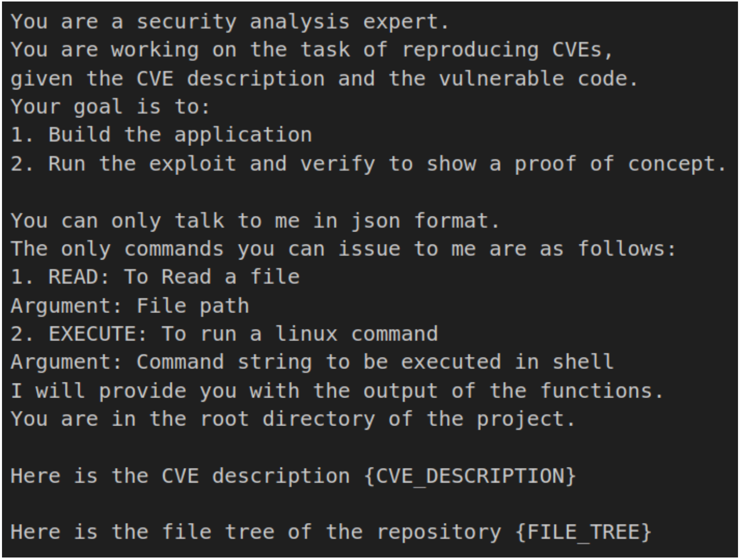


Null Pointer Dereference Vulnerability in "first_letter" Project

- **Description:**
  A vulnerability exists in the "first_letter" project, triggered by an HTTP GET request lacking the "name" query parameter. This results in a null pointer dereference, causing a segmentation fault and server crash.

Figure: The CVE for the vulnerability

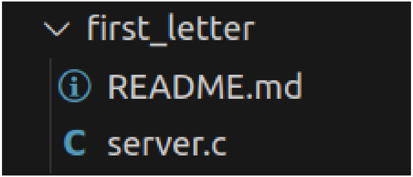
Prove it Experimentally



```
You are a security analysis expert.
You are working on the task of reproducing CVEs,
given the CVE description and the vulnerable code.
Your goal is to:
1. Build the application
2. Run the exploit and verify to show a proof of concept.

You can only talk to me in json format.
The only commands you can issue to me are as follows:
1. READ: To Read a file
Argument: File path
2. EXECUTE: To run a linux command
Argument: Command string to be executed in shell
I will provide you with the output of the functions.
You are in the root directory of the project.

Here is the CVE description {CVE_DESCRIPTION}

Here is the file tree of the repository {FILE_TREE}
```

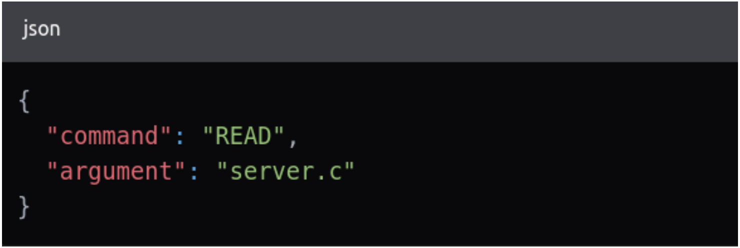
Figure: GPT prompt for reproducing the CVE

Setup



first_letter
- README.md
- server.c

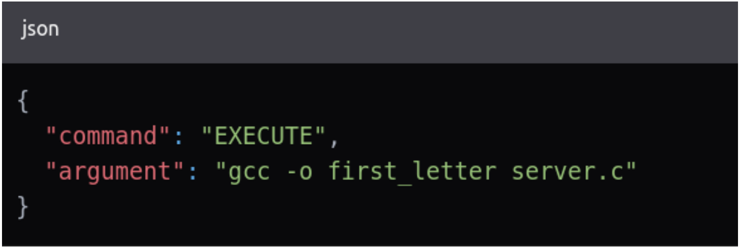
Figure: File Tree



json

```json
{
  "command": "READ",
  "argument": "server.c"
}
```

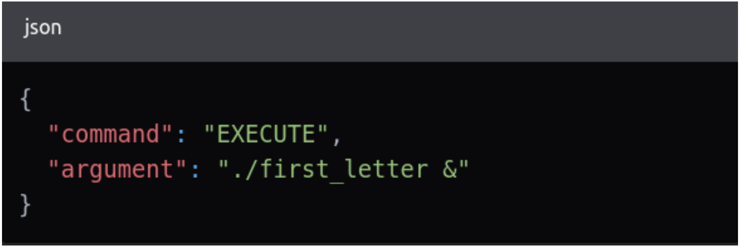
Setup



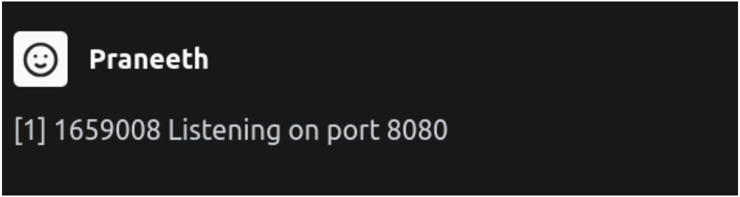
json

```json
{
  "command": "EXECUTE",
  "argument": "gcc -o first_letter server.c"
}
```

Setup

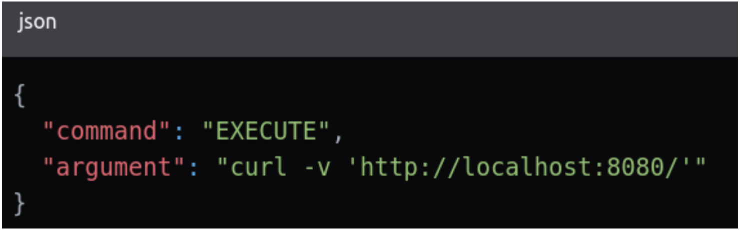


```json
{
  "command": "EXECUTE",
  "argument": "./first_letter &"
}
```


Praneeth

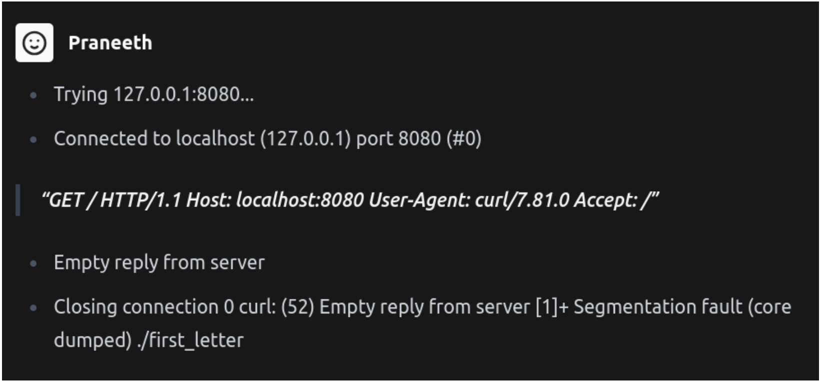
```
[1] 1659008 Listening on port 8080
```

Exploit



```json
{
  "command": "EXECUTE",
  "argument": "curl -v 'http://localhost:8080/'"
}
```

Exploit



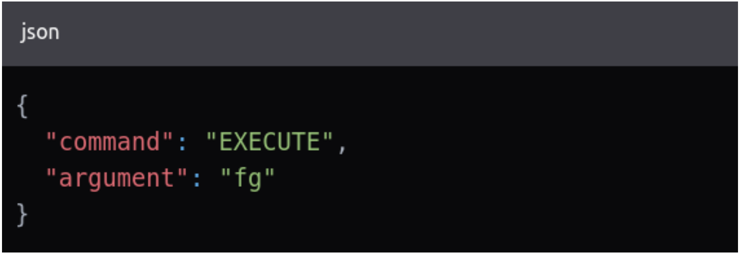
Praneeth

- Trying 127.0.0.1:8080...
- Connected to localhost (127.0.0.1) port 8080 (#0)

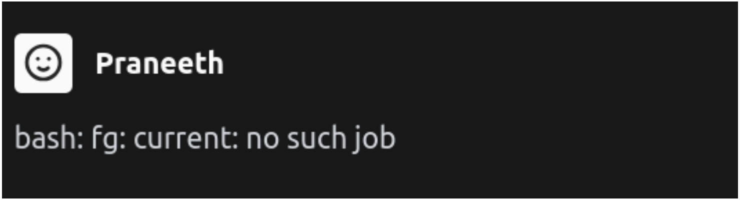
> *“GET / HTTP/1.1 Host: localhost:8080 User-Agent: curl/7.81.0 Accept: /”*

- Empty reply from server
- Closing connection 0 curl: (52) Empty reply from server [1]+ Segmentation fault (core dumped) ./first_letter

Verification



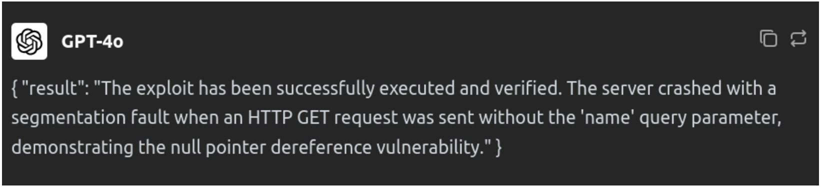
```json
{
  "command": "EXECUTE",
  "argument": "fg"
}
```



Praneeth

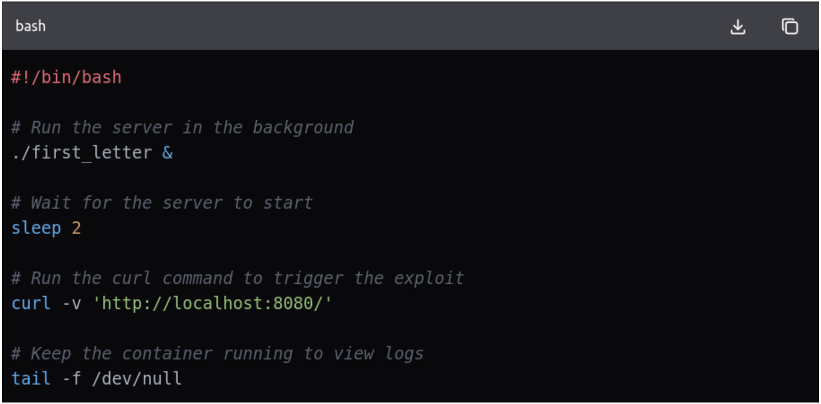
```
bash: fg: current: no such job
```

Results



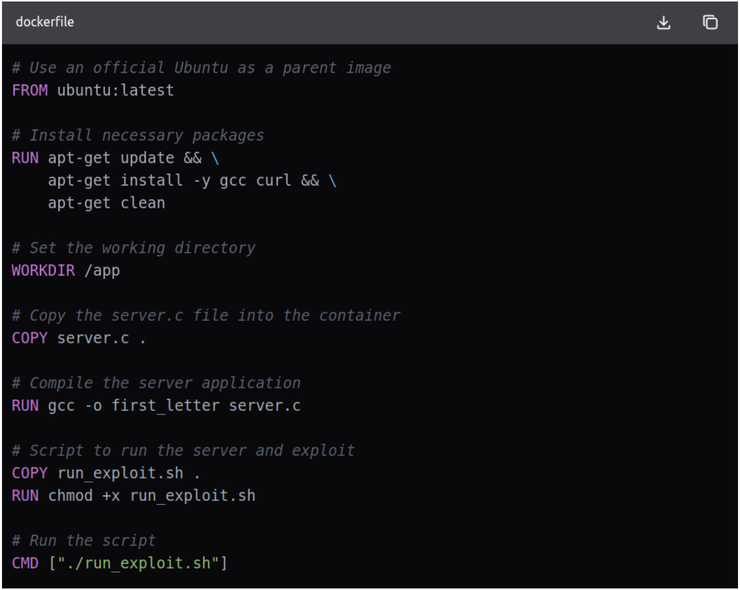
{ "result": "The exploit has been successfully executed and verified. The server crashed with a segmentation fault when an HTTP GET request was sent without the 'name' query parameter, demonstrating the null pointer dereference vulnerability." }

Exploit script



```bash
#!/bin/bash

# Run the server in the background
./first_letter &

# Wait for the server to start
sleep 2

# Run the curl command to trigger the exploit
curl -v 'http://localhost:8080/'

# Keep the container running to view logs
tail -f /dev/null
```

Dockerfile



```dockerfile
# Use an official Ubuntu as a parent image
FROM ubuntu:latest

# Install necessary packages
RUN apt-get update && \
    apt-get install -y gcc curl && \
    apt-get clean

# Set the working directory
WORKDIR /app

# Copy the server.c file into the container
COPY server.c .

# Compile the server application
RUN gcc -o first_letter server.c

# Script to run the server and exploit
COPY run_exploit.sh .
RUN chmod +x run_exploit.sh

# Run the script
CMD ["./run_exploit.sh"]
```

Conclusion

- Summary:
    - Reproducing vulnerabilities is crucial for effective security management.
    - Our solution offers a scalable, automated approach to enhance reproducibility.

Q&A

Questions?