Name: Neha Joshi

HW1: Report

The report is laid out in the following order:

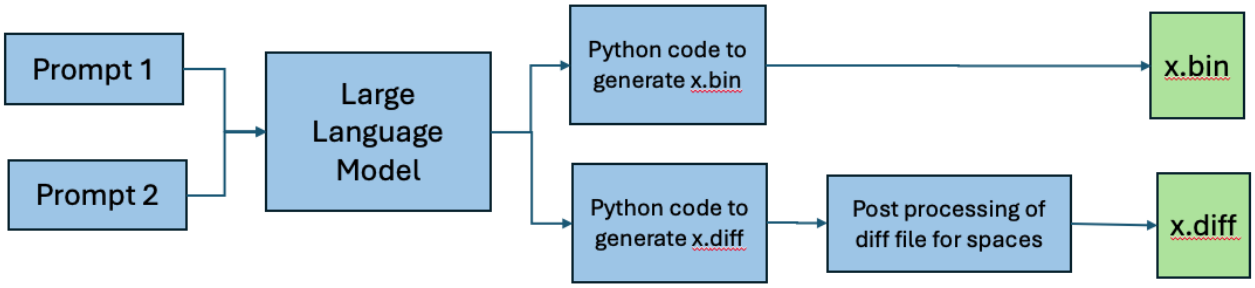
- Application features code architecture
- Details of LLM used
- Time and cost analysis
- Challenges faced and solutions
- Scope of improvement
- Instructions to run the code

1. Application architecture

The application is built upon a vulnerable code provided in mock-cp folder. This code has security vulnerabilities like a function in the code tries to read or write from memory that is private or inaccessible. This is a security threat because consider for example if passwords are stored starting from a memory location next to an accessible memory location. Accessing or reading them would lead to a great threat to the user whose password is read.

To solve all such issues posed by vulnerable codes, we design a Large Language Model based application which triggers a vulnerability in the code and then creates a patch from the same. Once, patched, the code is no longer vulnerable with respect to the patched vulnerability.

Now the following figure shows the architecture of the application and a details description follows:


Prompt 1
Prompt 2
Large Language Model
Python code to generate x.bin
Python code to generate x.diff
Post processing of diff file for spaces
x.bin
x.diff

The following steps are performed to design the architecture:

- First a prompt was written in multiple iterations which explains the LLM about the situation and asks to write a code that in turn generates a bin file which will trigger the vulnerability. (prompt 1)
- Next, another prompt was written separately that identifies the vulnerability and writes a modified code from the original code (referred from source). (prompt2)
- The code further creates a difference between original and modified file which is the diff file.
- The diff file has extra \n characters which are dynamically removed in the next step. This is not a hard coded step. It will just scan through the code and convert the \n into \\n to avoid any issues in the diff format. Finally the header of the diff file is verified and fixed.

2. Details of LLM used:

The current analysis uses Meta's Llama-3.1 mode which is among the top 3 models today.

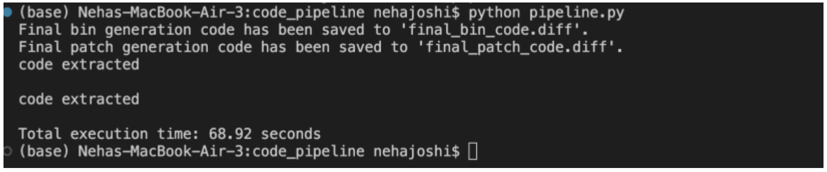
The prompts are written in a way that the model follows the instructions step wise. Temperature and max_tokens have been modified and tuned.

```
# Initialize client
client = openai.Client(base_url="http://127.0.0.1:11434/v1", api_key="EMPTY")
```

3. Time and cost Analysis:

TIME:

- The total time required to get the x.bin and x.diff file generated is: 68.92 secs
- I tried multiple ways to improve the model performance in terms of its answers like using different temperatures, running multiple iterations to get a constant improved code etc.
- However, the most optimized number of iteration was 1.



```
(base) Nehas-MacBook-Air-3:code_pipeline nehajoshi$ python pipeline.py
Final bin generation code has been saved to 'final_bin_code.diff'.
Final patch generation code has been saved to 'final_patch_code.diff'.
code extracted

code extracted

Total execution time: 68.92 seconds
(base) Nehas-MacBook-Air-3:code_pipeline nehajoshi$ 
```

COST:

- The cost in terms of number of tokens is as follows: (counted based on llama3.1 tokenization)
    - **Tokens used: 1700**
    - Prompt1 tokens: 285
    - Prompt2 tokens: 453
    - **Total Prompt tokens: 738**
    - Completion/ response 1 Tokens: 400
    - Completion/ response 2 Tokens: 562
    - **Total Completion/ response Tokens: 962**

4. Challenges faced and solutions:

- **Challenge:** The most challenging part of the question for me was to generate a valid patch! It took more than 50 iterations to get a working patch.
  Solution: Good prompt seems to be the only solution to this problem. I realized that the LLM needs to be told the tasks in a sequential manner one after the other. Completeness of the statement seems to be very important. Writing in caps or writing keywords like 'Important' don't seem to work.

- **Challenge:** Next related challenge for me was failed patching, corrupt patching and header issues in patch
  Solution: I first wrote a code in which I was telling the LLM the previous error and the patch that generated the error. However the LLM didn't seem to understand this. I also tried writing iterations giving previous code, patch and error but didn't work. Only solution here was writing a detailed and step wise prompt.

- **Challenge:** Next challenge was that the LLM was mentioning this task of triggering the vulnerability and getting a bin file for the same as illegal!
  Solution: Clearly wrote in the prompt that the task is for a sample and is totally legal. The LLM agreed to it. This might be an issue in general!

- **Challenge:** Getting a '\n' in the modified code for the patch. The modified code was a string which resulted in getting a new line in the patch! This was one of the biggest roadblock !!
  Solution: Wrote a post-processing step where all the '\n's as converted into '\\n' . This seamlessly created the right patch there after.

- **Challenge:** Setting the right temperature – Honestly my best result was on temperature 0.08 for one of the prompts. However, as the temperature was varied, I got different results each time and they varied a lot. Few even were totally incorrect codes. This created various issues in the pipeline.
  Solution: I kept the temperature 0 to get deterministic results.

- **Challenge:** Getting the pipeline setup.
  Solution: I faced issues in the pipeline setup as the LLMs have different outputs in each iteration. I solved this by dynamically extracting only the code portion of the answer using a regular expression.

- **Challenge:** Not getting the buffer overflow
  Solution: Initially the code written by the LLM was just ingesting NULL characters in x.bin. Solution was writing clearly in prompt that we need to write characters that lead to overflow.

5. **Future Scope/ challenges remaining:**

- I am still facing one issue that 1 in 10 times, the generated code is unexpected. While I know that LLMs work in a probabilistic way and even ChatGPT has given me wrong answers in general multiple times- I think hallucinations can be reduced by proper tuning.
- Due to the above I got all the commands mentioned in the HW pdf working but the last one failed in one of the runs.
- I think the prompt can be made shorter.

6. **Instructions to run the code:**

- Run the pipeline.py code
- Make sure the mock-cp folder is in the same folder as that of pipeline.py since I have added a path to the reference file as mock-cp/src/samples/mock_vp.c
- This is done to keep the application dynamic- we can just change this path and get the same patching work on a new code file.
- Get the x.bin and x.diff files.