Agentic CTF Solver Writeup

Authors: Yongjae Chung, Lucas Sylvester

Design Choices

Main Approach

As a starting point we used the basic command-line agent loop with two actions, "command" and "finish". We later expanded the range of available commands to include "python" and "write_file" to allow the agent to write payloads, scripts, and execute them. To attempt more approaches we iterated on two different agents, one using OpenAI models and one using Google Gemini. We did, however, find many common approaches that improved behavior for both of our models.

Adding a Thinking Step

We saw a notable improvement in the agent when it was instructed to perform an intermediate thinking step, making it reason about the next action it should perform. This reduced the times the agent was getting stuck in a loop of repeated commands. Below is a snippet from the `SYSTEM_PROMPT`.

```
You MUST always do internal reasoning (a THOUGHT step) before acting,
but you MUST NOT print or output this THOUGHT. Only output a single JSON object.
```

Similarly, for the Gemini agent, when given the option to not only do "command" and "final_message" but also "deliberate" once in a while, it seemed to both improve performance and make behavior more consistent.

LLM Utilization

Another approach in common between the Gemini and GPT agents was to include a deliberative-summary checkpoint in order to better interpret the logical flow of the agent in the process. For the GPT agent, this was implemented by adding an additional call to the LLM every `SUMMARY_CHECKPOINT = 5` steps; summarizing the actions performed so far, reminding the agent about main goals, and planning the next actions.

Edge Cases

For both agents, we had several edge cases in common.

The first was managing the agent's permissions for problem 1. Since the agent was not allowed to read the Python script that contained the `eval` vulnerability, we instructed the agent to explicitly try arbitrary execution of code through the input of the program, providing an initial approach for these kinds of challenges.

Another issue was the location of the `flag.txt`. Since the vulnerable binaries contained the path `/flag.txt`, our agents would run into this error: `"cat: /flag.txt not found"` even though the vulnerability had been successfully exploited. To troubleshoot this, we used prompting to instruct the agents to relocate the `flag.txt` to the root level when encountering this error.

For the GPT agent specifically, it sometimes attempted to remotely execute an exploit with pwntools, using the network details in the challenge.json. This behavior resulted in additional unnecessary steps, wasting the step budget. Since our challenge was required to be solved locally, we prompted the agents to refrain from these attempts. The resulting agent was able to arrive at a solution more quickly on average without doing these unnecessary steps.

Limitations

Both agents had problems with repeated commands.

For the GPT model this resulted in repeated commands, and similarly for the Gemini model it seemed to get stuck in long repeating strings, wasting the token budget.

The Gemini model, when hitting a max token budget, would begin returning empty strings instead.

While both agents could consistently solve the three given challenges, they both struggled when run on new pwn challenges from previous CSAW competition years.

Agents

Gemini Agent: https://github.com/luca2618/ML-CyberSec-2025-Lab2-Public
GPT Agent: https://github.com/yongjae354/ML-CyberSec-2025-Lab2/blob/main/gpt-agent.py