Effectiveness of Large Language Models for Incident Response on Windows Operating Systems

Daniel N. Ho*, Matthew J. Ho**

*South Sutter Charter School, **Franklin High School

Introduction

With the ever-increasing use of large language models (LLMs) in everyday usage, there is an increasing interest in the use of LLMs in the field of cybersecurity. This project aims to investigate the effectiveness of large language models when it comes to incident response in the aftermath of cyber attacks, specifically in the scope of Windows operating systems. Our goal is to evaluate the capabilities of different LLMs as automated tools for the incident response and remediation on Windows operating systems, specifically in cases where the response to a cyber attack on multiple independent systems must be done quickly and effectively.

Incident response is a critical part of the field of cybersecurity, in which the detecting, analyzing, and remediation of cybersecurity attacks is a constant threat in today's ever so digitalized world. However with the increasing amount of data passing through digitized servers and systems, it becomes progressively more resource intensive for humans to effectively identify attacks and remediate vulnerabilities manually.

This project aims to evaluate the incident response capabilities of several LLMs on Windows operating systems in order to gauge potential effectiveness in production environments.

Related Work

LLMs for offensive cybersecurity. The Cybench[1] project evaluates LLMs on CTF challenges, which are cybersecurity related problems created for the purpose of promoting cybersecurity. Through the testing that Cybench has done, they have shown how effective artificial intelligence can be in problem solving real-world cybersecurity problems. This examination of the ability of large language models to problem-solve through complex situations is a point that we want to explore through the response to the often intricate cybersecurity attacks that occur every day. Other projects, such as BountyBench[2], CVEBench[3], Incalmo[4], and XBOW[5], test the offensive cyber capabilities of LLMs using recently discovered vulnerabilities, realistic corporate networks, and real bug bounty programs.

LLMs for patching vulnerabilities. The BountyBench[2] project is a framework that evaluates LLMs on detecting, exploiting, and patching vulnerabilities in real-world open source codebases. It differs from some other LLM cyber benchmarking projects in that it not only evaluates exploitation ability, but also patching vulnerabilities. Through testing of several different artificial intelligence agents, they found that such agents are notably effective in writing vulnerability patches. We wish to extend this research by using LLMs to patch vulnerabilities on live operating systems rather than just static codebases.

Methodology

Overall Experiment

In order to test the capabilities of LLMs in remediating vulnerabilities on a Windows machine, we created a scenario in which a machine has been successfully infiltrated by an attacker. The attacker has installed persistence on the machine and altered the system's settings, leaving behind evidence of exploitation. We evaluated two models (GPT-5 Codex and Claude Sonnet 4.5) based on how well they were able to identify the attacker's actions and resolve any discovered vulnerabilities.

Agent Architecture

We provided the LLMs access to the Windows machine through the Codex CLI[6], an open-source coding agent that can run directly in a terminal and issue commands. We ran the Codex instance on Windows Subsystem for Linux (WSL), where it was able to both access the whole Windows filesystem and execute Powershell commands.

To evaluate how well the agents remediated vulnerabilities, we utilized the open-source project Aeacus[7], used in cybersecurity defense competitions to evaluate the presence of vulnerabilities on a system. After the LLMs have finished their work on the machine, we used Aeacus to objectively tell whether or not vulnerabilities were remediated.

We collected data about all commands run by the agent (with timestamps), as well as an LLM-generated description of each action taken. In addition to this, we had the models answer questions about what the attacker did to get initial access, what the attacker did after gaining access, and what was done to remediate these actions.

More information about the agent architecture is in Figure 1.

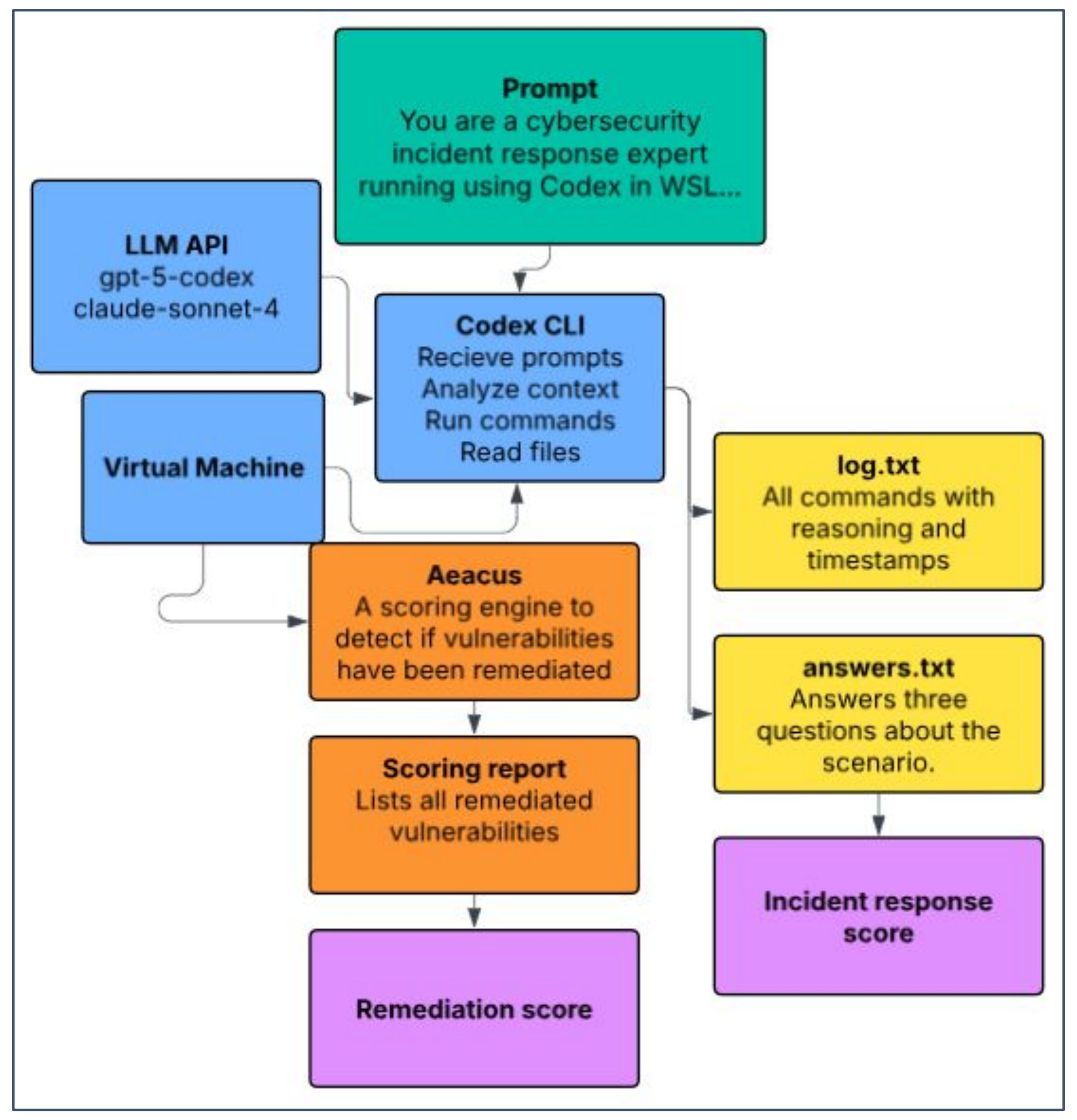
Prompt
You are a cybersecurity incident response expert running using Codex in WSL...
LLM API
gpt-5-codex
claude-sonnet-4
Codex CLI
Recieve prompts
Analyze context
Run commands
Read files
Virtual Machine
log.txt
All commands with reasoning and timestamps
Aeacus
A scoring engine to detect if vulnerabilities have been remediated
answers.txt
Answers three questions about the scenario.
Scoring report
Lists all remediated vulnerabilities
Incident response score
Remediation score

Figure 1: Agent Architecture

Scenario Architecture

We provided LLMs with a Windows 10 machine with the following:

- An exposed remote desktop service without network level authentication enabled.
- A sticky keys backdoor allowing for privilege escalation.
- An insecurely set disabled Tamper Protection setting that allowed for interference with Windows Defender.

To simulate exploitation of the environment, we performed following additional actions on the machine:

- Connected to the machine through the open remote desktop unauthenticated.
- Utilized the sticky keys backdoor to create a privilege escalated account with full administrator permissions.
- Used the newly made account to turn off Windows Defender, turn off the firewall, and install a netcat reverse shell for persistence.

Results

We evaluated the LLMs based on the number of vulnerabilities they patched, as well as how long they took to patch those vulnerabilities. The two runs with GPT-5 Codex found 2 and 3 vulnerabilities, and the two runs with Claude Sonnet 4.5 found 3 and 5 vulnerabilities (Figure 2). The GPT models were generally slower to remediate vulnerabilities than the Claude runs, with no vulnerabilities found in either run before 15 minutes, while Claude found all its vulnerabilities before 10 minutes (Figure 3).

The netcat reverse shell was the easiest vulnerability to find, found by all of the four runs. No runs fixed the sticky keys backdoor.

Run	reverse shell	malicious user	disable RDP	sticky keys	firewall	defender misconfig
GPT 1	00:17	N/A	N/A	N/A	00:18	N/A
GPT 2	00:17	00:19	00:19	N/A	N/A	N/A
Claude 1	00:04	00:03	N/A	N/A	00:06	N/A
Claude 2	00:04	00:03	00:06	N/A	00:06	00:07

Figure 2: Time (in minutes) that models took to remediate vulnerabilities

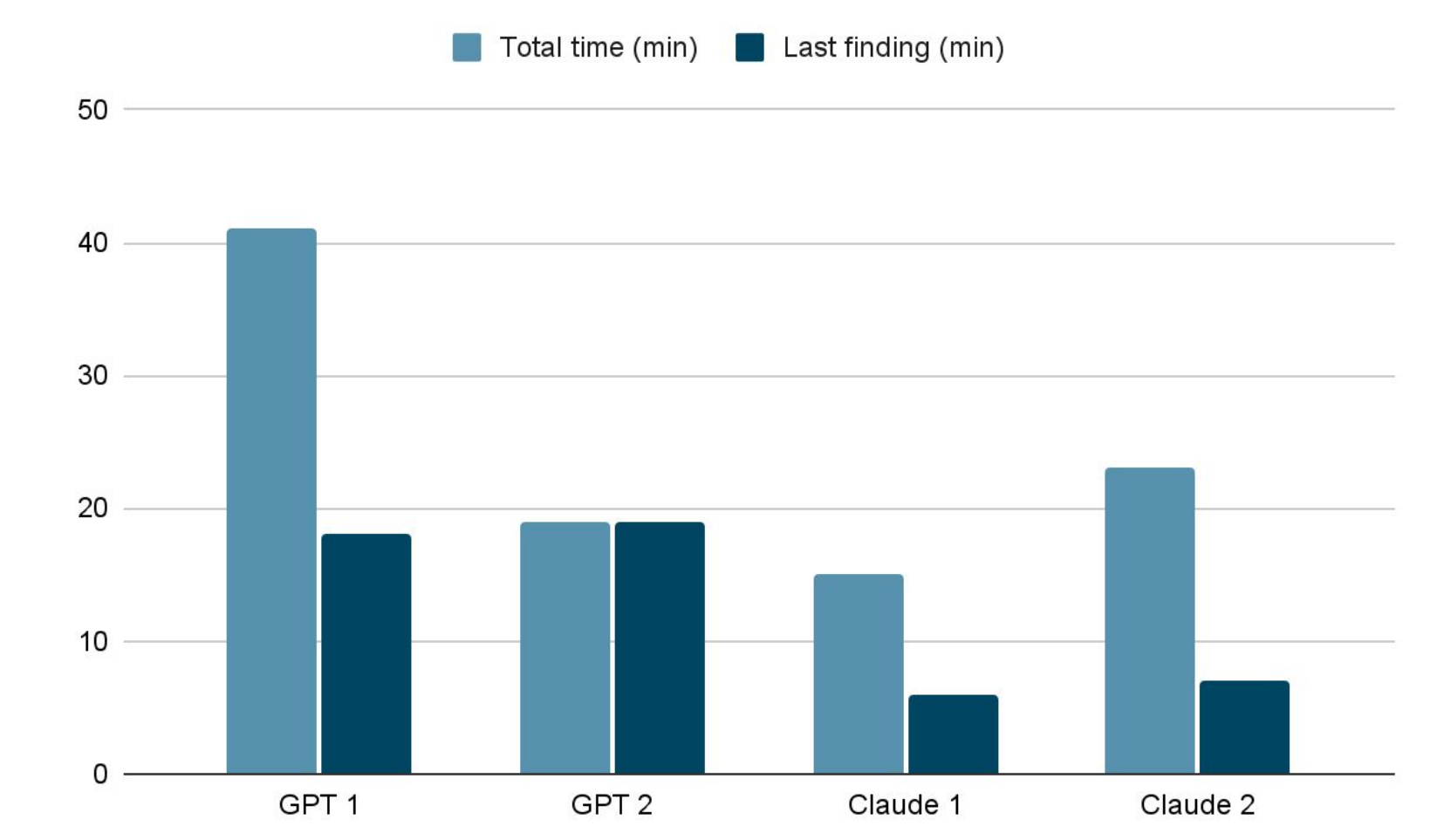

Figure 3: Total running time and time of last vulnerability remediation (in minutes) for each run

Conclusions

We found that both models tested in our study were thorough in their reasoning process and analysis. Both were able to find certain vulnerabilities. However, GPT-5 Codex was much slower than the Claude Sonnet 4.5 runs, even though both models found vulnerabilities at around the same speed. This was because it tended to perform unnecessary (and time-expensive) work, such as performing string searches through the whole filesystem. This search would usually time out, and only after this would GPT actually fix vulnerabilities. Claude would find a vulnerability and immediately fix it, and when it scanned for more, it would have more reasonable sized scans. This is likely the reason Claude was able to find more vulnerabilities than GPT was, in a shorter amount of time.

We conclude that Claude Sonnet 4.5 appears to be more effective at performing incident response and remediation than GPT-5 Codex, because of its better ability to manage time and complexity tradeoffs.

Discussion

Although this project does provide much insight into the ability of LLMs to perform incident response on Windows machines, its scope could be improved to more accurately benchmark LLM incident response capabilities. This could be done in several ways:

- Utilizing additional LLMs, rather than just GPT-5 Codex and Claude Sonnet 4.5.
- Performing more runs with each model, to limit the effect of the inherent non-determinism of LLMs.
- Creating more scenarios to test a more diverse range of incident response capabilities.
- Provide graphical user interface (GUI) access to the agent for further LLM capabilities.
- Improve time management through methods other than innate LLM characteristics, such as prompting

Future work can be done to improve upon this study, and to make it a more comprehensive benchmark of LLM performance in incident response tasks.

References

[1] Andy K. Zhang, et al, "Cybench: A Framework for Evaluating Cybersecurity Capabilities and Risks of Language Models," 2025.

[2] Andy K. Zhang, et al, "BountyBench: Dollar Impact of AI Agent Attackers and Defenders on Real-World Cybersecurity Systems," 2025.

[3] Yuxuan Zhu, et al, "CVE-Bench: A Benchmark for AI Agents' Ability to Exploit Real-World Web Application Vulnerabilities," 2025.

[4] Brian Singer, et al, "On the Feasibility of Using LLMs to Autonomously Execute Multi-host Network Attacks," 2025.

[5] Xbow. "Xbow Homepage." https://xbow.com/

[6] OpenAI. "Codex CLI Documentation." https://developers.openai.com/codex/cli/.

[7] Elysium Suite. "Aeacus." https://github.com/elysium-suite/aeacus.