Automation of Cyber Exploitation through AI

Matthew Zehnder
Department of Cyber Science
U.S. Naval Academy
Annapolis, MD, USA
m267146@usna.edu

***Abstract*—AI has been an ever-evolving field that has found its place in every part of tech, from Google's searches with Gemini, to Ai-run help chats for private companies. With Ai being incorporated in all these different places the dangers of Ai need to be explored further. This project serves to help further understand locally-run Ai models' abilities to reason through simulated cyber attacks. This research uses Hack the Box(HTB) capture the flags(CTF) to simulate cyber attacks that the AI may be able to solve. While these are not true cyber attacks, it helps to simulate the AI's ability to reason about the cyber attack process in many environments where different vulnerabilities are available. While still in the testing phase, the current results are promising. The Ai frequently passes the first stages of the CTF; showing a developed understanding of how to find where the vulnerability may be and what initial command it should use. When attempting multi-step attacks, the agent often becomes confused, repeating itself or refusing to give a command. These findings show a glimpse of what Ai can and will be capable of within the cyber domain. With this research's position of not aiding the AI in the cyber threat, it is still able to make it past the first stage of the CTF (cyber attack). With additional help or future smarter models with better reasoning, AIs may be able to reach a stage of end to end cyber threats soon.**

I. Introduction

With AI's reasoning improving at such a rapid rate in the realm of small, locally run, LLMs this research attempts to start answering the question of: "are smaller run local LLMs capable of executing cyber exploitation?" The implication of this research is to figure to what extent can a small scale threat use LLMs to complete cyber exploitation with minimal knowledge on the target, only ip. This tests the models' abilities to reason through cyber exploitation based solely on the knowledge it already possesses. Using a self feeding loop, the AI is expected to learn about the target from the output of the terminal as it executes code. These cyber exploitation tests were all conducted on Hack the Box(HTB) capture the flags(CTFs) in order to ensure a safe environment to test the AI's abilities.

This is not a new idea; the use of AI, in cyber exploitation, has been tested before. In terms of using the MITRE ATT&K framework, AI has been tested in its ability to produce effective executable code for cyber-attacks at a success rate of 16% [1]. This was done when the AI had no prior knowledge of the machines it was attacking and the full attack was created before execution, in a script style. Self feeding AI threat loop has also been tested in the form of automatic pen testing [2]. Here, Chatgpt3.5 was used to execute privilege escalation on virtual machines it was sshed into. The goal with this approach is to get a more widespread view of what locally run LLM can reason through, attempting to see what exploits they can identify and execute.

II. Methods

Two primary components to the methods used cover the code itself and the application of the code onto CTFs. The code aspect covers the construction of **ai_implication.py** including the components that translate the AI's reasoning to executable code. The flow is the larger scope of the code indicating how the code interacts with the command line and the methods the AI used to solve the CTFs.

A. Code components

In order to interact and use the AIs Ollama was used. Ollama is a Rust based program that allows for the pulling(downloading) of open source AI models, to be run on local machines including libraries for interaction within python code [3]. The two different AI models were pulled to run against ctfs; initially llama 3.1:8B, one of Meta Ai's premiere small AI's and QwQ-32B, a newly released Ai meant to push the boundary of what small locally run Ai models could do. The models were both chosen for two main reasons: built-in tool functions, allowing someone to describe the type of response that is desired to get a more consistent response (ie. commands) and their smaller size to be able to be run locally.

QwQ takes into account 32 billion parameters when running generation. This results in a sophisticated level of reasoning when compared to other small scale LLMs at a cost of a larger weight on the system. The larger weight comes in the shape of requiring more VRAM/RAM as well as computing power to run, resulting in a longer generation time when compared to smaller models. QwQ was created to leverage advanced reasoning and critical thinking techniques to produce better results downstream, especially in complex problem solving [4]. This ability to excel at complex problem solving ties well into CTF application, due to CTFs high level of complexity.

Llama 3.1:8B only takes 8 billion parameters into account during its reasoning phase. The fewer parameters result in lessened reasoning abilities when compared to QwQ, but a much faster run time. This smaller weight allowed for llama 3.1 to be run drastically faster than QwQ which allowed for much quicker and easier testing using llama 3.1. Llama 3.1 was created for the purpose of implementing tool calls to use alongside its coding skills and multilingual abilities [5]. This

main purpose for llama 3.1 did not play as well into solving the CTFs as QwQ did.

- The message stream is the object that is used to query the AIs and keep their memory. The new query was appended to the end of the message stream containing all the info from previous queries. This gave the AI the ability to use the new information in tandem with previous responses to reason through different options, producing new outcomes.
- **tool_calls** were the backbone of getting consistent executable commands from the AI, giving the ability to describe the desired responses. The descriptions resulted in the models delivering more fitting responses. The **tools_calls** pointed to one tool "extract_code" containing three different parts: "code_type" used to have the Ai interpret they type of code it thinks it is extracting, "code" the actual code that the model is producing to be executed on the command line, and "why" the reasoning behind why the Ai choose this code.
- **Tmux** calls were used to run the commands in the terminal. By using **Tmux** optimal supervision of command execution live was achieved, additionally, it offered the ability to step in if the command did not execute for unforeseen reasons. This helped in cases such as uninstalled tools in which case the tools could be downloaded manually for the AI to use, allowing for more efficient testing.
- Unique csv files were created with each instance of the code. These csv files acted as logs for the CTF attempts and later analyzed for patterns and habits of the AIs.

B. Apply the code to CTFs

The information about the CTF was fed into the code in the shape of a target IP, as seen in Figure 1 below. Initially the LLMs only got the IP address of the target giving it minimal outside input on how to approach the problem. By giving it only basic information the AI has to choose its own approach of attack: including, when it would prefer to find out information and when it would rather than try attacks. The recursive nature of the program's logical execution tests the AI's ability to reason with its own results. This helps to see how the AI chooses to traverse the MITRE ATT&CK framework, by giving it the freedom to choose whichever path it wants to attack the target IP.

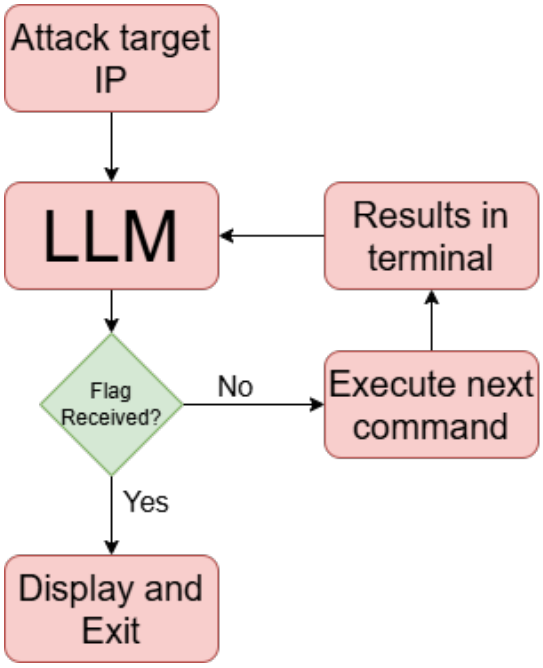

Fig. 1. Flow Chart depicting program's logical execution. A recursive feedback loop between the LLM and the command line.

The code was then run against 14 different CTF's from HTB. Most of the CTF's were under the easy or very easy category, building the stepping stones for what the AI could do. Each test took up to 30 minutes to complete. This is because of the long reasoning times of QwQ that came from hardware restrictions. Depending on the AI's initial success of the CTF, retests of the CTF were conducted. Some of the retests were due to the AI becoming trapped by looped nonsense, getting nowhere in the CTF. In these sprees of looped nonsense the AI would refuse to find new commands, in an attempt to force the use of a command it thinks should work.

Initially when running the test, I/O blocking was used to know when the command line completed execution. This allowed for more rapid tests when everything ran smoothly, but was often unsuccessful, especially when the command would enter a new client resulting in the need for more input before the I/O blocking cleared. This issue resulted in more problems than aids as it made clients unusable by the A,I which were oftentimes the heart of the CTFs. The alternative which was manual continuation of the code was used which also allowed for live aid to the AI: such as when the program queried a command that was not yet installed, allowing for a manual installation of the program to make the attempts run more efficiently.

III. Results

The preliminary results were promising, showing the Ai was able to quickly and easily complete the beginning steps of solving the CTFs. It consistently was able to find the avenue to the vulnerability in the CTF, as well as knowing the initial commands to start interacting with said avenue. This was seen as nearly every time QwQ was able to successfully use

nmap to find out more information on the IP and where the vulnerability was.

Furthermore, the AI's did not excel at going further into the CTF. The AI got lost when using more specific tools, after the initial phase of gaining contact to the location of where the vulnerability was. While the AI knew what tools to use, it was unable to effectively use them, especially with more niche clients such as **redis-cli**. Once in these clients QwQ would often either give up not responding with a tool call, killing the program, or it would provide non-working commands repeatedly.

A. QwQ vs Llama 3.1

The largest distinguishing feature in the models was the parameter sizes, with llama 3.1 at 8 billion parameters and QwQ at 32 billion parameters. This results in a much more sophisticated reasoning from QwQ at the cost of much more computer power. The effects of needing additional power and run time is depicted in the graph below in Figure 2. On the hardware side llama 3.1 only required around 8gbs of space in VRAM/RAM to run allowing the whole model to be loaded easily. QwQ required around 24gbs of space in VRAM/RAM to load the model onto in order to run the model. This drastic difference between the two models lead to very different results from the reasoning.

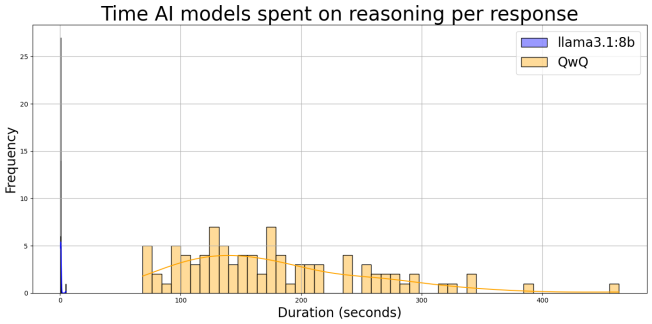

Fig. 2. A histogram depicting the time spent reasoning in seconds of both QwQ and llama 3.1 [6].

Even though QwQ took substantially longer to run, the value of the reasoning ability overcame its timely downside. Within the actual CTF, the different reasoning abilities between the two can be seen; QwQ was able to reason through **nmap** not working due to incorrect options, while llama 3.1 repeated the same command multiple times, often unable to produce new usable commands. This shows QwQ's higher level understanding of both what the **tool_calls** wanted as well as the structure of the question at hand. Smaller LLMs can still find their place in cyber exploitation. Using AIs of different sizes provides advantage in areas that do not require a lot of reasoning but do require lower latency [7]. With varying needs, varying AIs can be used in order to optimize performance.

Initial testing was completed all using llama 3.1 for two reasons. First, QwQ was not out yet (QwQ was released in mid March of 2025 as a cutting edge smaller scale LLM), Second, better hardware became available allowing for the use of larger LLM to be run locally. QwQ being released so recently shows how quickly research in the AI and computer domain can change.

IV. Discussion

What the results show is a low level ability for AIs to understand how to go about solving CTF, including the initial steps to take. But, getting stuck further within the CTF shows the AIs lack of understanding the more sophisticated sides of the CTFs.

A. Current viability for threat use

Even though the AIs were unable to complete exploitation the results show even now AIs could aid in cyber exploitation. A possibility, viewing from the scope of being used in mass attacks as an initial contact strategy. By giving an AI a bank of IPs to be run through, it could potentially deliver possibly vulnerable IPs to an adversary. While not being a sophisticated cyber cracker the AI has shown itself as a reasonable cyber exploit sniffer, with the ability to be run 24/7 it could be seen as a possible threat.

Against the argument of better current cyber exploit sniffers already being available, the use of AI in this frame provides a unique quality most code can not. The AI's inherent ability to adapt to and overcome challenges using reasoning vice procedure, allow AI in this position to possibly get past new problems unseen before. This ability is valuable, with new vulnerabilities continuing to rise each year [8]. Having the advantage of reasoning in combination with the ability to run endlessly creates an adaptive exploit sniffer able to detect vulnerable IPs non stop.

B. Movement in the MITRE ATT&CK framework

The MITRE ATT&CK is a widely used benchmark when it comes to analysing cyber threats and adversaries.The framework is a database containing malicious activity from real world incidents to be used for further forensic and behavior analysis of an adversary [9]. This database provides a way to depict success and progress within an exploitation by giving each stage a tangible definition.

QwQ showed a diverse intent to traverse the mitre ATT&CK Framework. These attempts show the AI is not limited by one field of thought when approaching a CTF. Figure 3 below shows a weighted nodal representation of concepts and methods that the AI tried to use when approaching the CTF. The graph shows the AI's ability to move forward through the same steps framework using different methods, showing reasoning ability to discover the best method instead of relying on one way to do the exploitation.

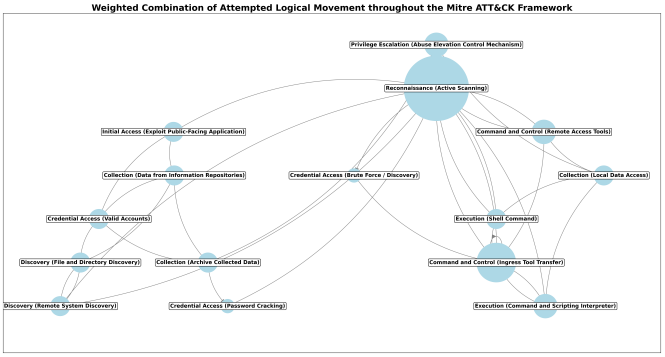

Fig. 3. A weighted nodal map of attempted movement through the MITRE ATT&CK framework based on commands associated with each stage [6].

Figure 3 above was created using a unique dictionary that corresponded commands used within the tests to aspects of the MITRE ATT&CK framework. This unique dictionary was then transponded onto the graph including the links between conceptual attempts and weighting of each node based on use.

These results of traversing were also heavily influenced by the CTFs chosen. Each CTF is curated to specific vulnerabilities, in a way providing a predetermined path. Even with the predetermined path QwQ was able to follow most of the paths to the location of the vulnerability. If the path was less clear as in the real world, there may be different results.

C. Self Correcting Ability

QwQ showed an ability to self-correct its commands as seen in Figure 4 below. Oftentimes, the model would get the original command wrong, forgetting an argument or similar ailment, and then would correct itself based on what the response was from the command line. This goes to show the model's ability to reason with itself, adding a persistent factor that is not available without AI. This leads into heavier implications, such as how well an AI may fare when it is met with resistance in a cyber attack. The ability to quickly learn and adapt within the CTF process shows promise regarding AI's future in cyber threats.

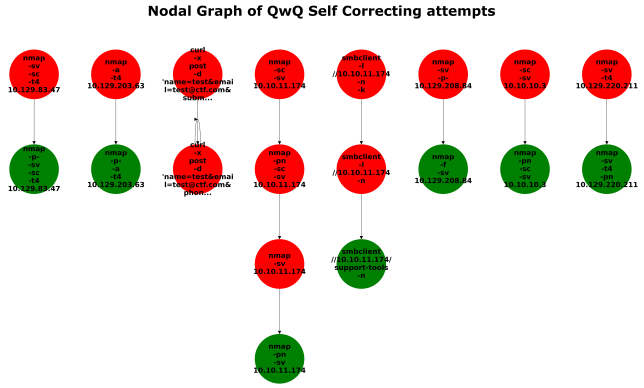

Fig. 4. . A nodal graph showing QwQ attempts at self correcting. This was captured by finding similar succeeding commands and seeing if they moved on successfully [6].

This leads into heavier implications, such as how well an AI may fare when it is met with resistance in a cyber attack. By using the ability to rewrite commands when they are wrong brings in a human element to the table.

Although self correcting was not always the case. There were times QwQ experienced hallucinations at the start of an exploitation attempt. Hallucinations are when a LLM produces incorrect or nonsensical information or in this case commands [10]. When given the initial IP there were times when QwQ would give unfavorable responses to the system leading to compounded hyper specific commands. From an non-executable state the AI would choose to add onto the command vice, finding a new avenue.

D. Set Backs

The biggest setbacks in this research was dealing with technical issues. When working with locally run AI's on non high performance computers they run slowly and need further configuration. This led to difficulties in the early stages of testing, as the debugging process required the AI to run for each attempt slowing down the process greatly. These technical issues helped to play into the realm what an adversary may encounter when inputting such strategies. As this research points to what a low level adversary may be capable of.

E. AI Cooperation

A major expected issue was the Ai refusing to produce commands due to their inherent malice of the commands needed. Surprisingly this was not an issue at all. The only gearing done to allow this would be the fact the mission was to solve a CTF, but the AI never second guessed producing malice code after that. The only refusal of code came from the AI not using the **tool_calls** when it became confused, but malicious code such as **gobuster** or **zipcracker** were produced without complaint.

Due to the AI models unquestioning nature to provide inherently malicious code the question of possible dangers from the AI arise. AI's adherence to providing a working response leads to concerns over the ability to use AI in malicious ways [11]. It seems that convenience and usability has overshadowed safety in the process of creating AIs. Heavier safeguard should be put into place, solely relying on the user to claim use is for good is not enough.

V. Future

Looking into the future more tests need to constantly be conducted on AI as it is always evolving. AI is still growing at a rapid rate continuing exponential growth similar to Moore's Law [12]. So it is important to continue to research various aspects of AI, especially the possible harm that they can cause. In terms of exploring cyber exploitation there are a few obvious avenues to approach.

A. Continuity

Further research using more powerful locally run AI on better hardware. This will help to see how much better larger

local LLMs are at reasoning through simulated cyber attacks. Similar to 2023 study on automatic AI pentesting using chat GPT3.5 where they used AI to pentest privilege escalation on devices [2]. With AIs evolving at the rapid rate they currently are, this testing can be perpetual as new more powerful AI comes to light.

B. Embedded Information

Test AIs ability using embedded databases with CTF information. Using an embedded database such as **chromadb** will allow the AI model to reference a database full of applicable commands in tandem with the AI's reasoning. This dynamic system would allow for quick and easy testing of different data methods, without the need for heavy commitment to one method. Opening the possibility of a live database allowing current updated methods to be applied as Cyber threats.

C. Specialized Model Training

A more resource heavy future test would be training an Ai model on various CTF materials. Using objects such as CTF walkthroughs, metasploit tutorial, and CTF cheat sheets can allow for the Ai itself to be more in-tune with cyber threat capabilities. This method would come at a heavy cost as training AI's is a costly and time consuming activity. But, if an AI were to be trained in this fashion it may be able to adapt to cyber threats faster and more efficiently.

REFERENCES

[1] E. Iturbe, O. Llorente-Vazquez, A. Rego, E. Rios, and N. Toledo, "Unleashing offensive artificial intelligence: Automated attack technique code generation," *Computers & Security*, vol. 147, p. 104077, 2024.

[2] A. Happe and J. Cito, "Getting pwn'd by ai: Penetration testing with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 2082–2086.

[3] H. Lin and S. Tawab, "ollamar: An r package for running large language models," *Journal of Open Source Software*, jan 2025. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.07211

[4] Q. Team, "Qwq-32b: Embracing the power of reinforcement learning," March 2025. [Online]. Available: https://qwenlm.github.io/blog/qwq-32b/

[5] M. AI, "Llama 3.1 8b instruct," https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct, 2024, accessed: 2025-04-28.

[6] OpenAI, "Chatgpt: Optimizing language models for dialogue," https://openai.com/blog/chatgpt, 2022, accessed: 2025-04-28.

[7] J. Mahapatra and U. Garain, "Impact of model size on fine-tuned llm performance in data-to-text generation: A state-of-the-art investigation," *arXiv preprint arXiv:2407.14088*, 2024.

[8] K. Bennouk, N. Ait Aali, Y. El Bouzekri El Idrissi, B. Sebai, A. Z. Faroukhi, and D. Mahouachi, "A comprehensive review and assessment of cybersecurity vulnerability detection methodologies," *Journal of Cybersecurity and Privacy*, vol. 4, no. 4, pp. 853–908, 2024.

[9] B. Al-Sada, A. Sadighian, and G. Oligeri, "Mitre att&ck: State of the art and way forward," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–37, 2024.

[10] S. A. Athaluri, S. V. Manthena, V. K. M. Kesapragada, V. Yarlagadda, T. Dave, and R. T. S. Duddumpudi, "Exploring the boundaries of reality: investigating the phenomenon of artificial intelligence hallucination in scientific writing through chatgpt references," *Cureus*, vol. 15, no. 4, 2023.

[11] D. Glukhov, I. Shumailov, Y. Gal, N. Papernot, and V. Papyan, "Llm censorship: A machine learning challenge or a computer security problem?" *arXiv preprint arXiv:2307.10719*, 2023.

[12] M. R. Douglas and S. Verstyuk, "Progress in artificial intelligence and its determinants," *arXiv preprint arXiv:2501.17894*, 2025.