International Journal of Scientific Research & Engineering Trends
Volume 11, Issue 5, Seot-Oct-2025, ISSN (Online): 2395-566X

Autonomous Penetration Testing with Cyberguardian: A Large Language Model-Based Approach

Jeslin Hashly, Jestin K Sunil, Alby Shinoj
Division of computer science and Engineering
Karunya institute of technology and Sciences, Coimbatore, India

Abstract- This paper introduces CyberGuardian, a new LLM-based agent for autonomous penetration testing. CyberGuardian is composed of two parts: a planner and a summarizer. These parts cooperate to create and carry out commands in an iterative manner. To evaluate CyberGuardian, we introduce two new benchmark suites based on the popular Capture the Flag (CTF) systems PicoCTF and OverTheWire, comprising 200 challenges in various domains and levels of difficulty. Our experiments check CyberGuardian's most critical parameters, such as levels of creativity and token usage, on LLM. Results reaffirm the need of good security procedures and show how LLM-based agents can advance autonomous penetration monitoring.

Index Terms- Cybersecurity Automation, Large Language Models, Benchmarks.

I. INTRODUCTION

Highly scalable and robust cybersecurity solutions are necessitated by the high rate of growth in the amount and sophistication of cyber threats. Traditional penetration testing with heavy reliance on human intuition is no longer viable with more and more sophisticated systems. Automation of penetration testing has been a practical solution to reduce these problems. Recent developments in large language models (LLMs) have expanded the potential for more adaptable and intelligent systems. The promise of LLMs in cybersecurity is demonstrated by CyberGuardian, an LLM-based agent that can resolve CTF problems independently.

Penetration testing is necessary in order to find and neutralize vulnerabilities via the simulation of cyber-attacks against systems. As systems continue to grow in complexity and the potential for vulnerabilities, manual penetration testing is becoming increasingly resource-based. Penetration testing automation has also become an appealing choice in addressing scalability and efficiency issues. Heuristic-based tools have been widely accepted and offer automated scanning and vulnerability detection. But these technologies lack the flexibility and fine-grained problem-solving ability that is required to handle advanced or novel security issues.

Recent advances in LLMs have demonstrated exceptional capability in understanding and generating text similar to a human, revealing new avenues to their application within cybersecurity. Utilizing LLMs in penetration testing introduces the potential for faster and more intelligent systems. Prior tests, such as PentestGPT or HackingBuddyGPT, have promised promising results using LLMs to assist in penetration testing activities. These programs continue to require human operators for some functions, limiting their scope and autonomy.

II. PREVIOUS WORKS

Recent advancements in the utilization of Large Language Models (LLMs) towards cyber defense have shown tremendous potential towards automating most of the tasks involved in penetration testing. PentestGPT and HackingBuddyGPT are good pointers towards the high level of results possible through the application of LLMs in assisting in penetration testing processes. These systems, however, require human operators to execute certain actions, such as commands or interface interactions, which limit them to complete autonomy and scalability. AutoAttacker, a notable addition, exploits automatically but is limited to using the Metasploit framework, reducing its applicability in certain types of hacking. Enigma, the cutting edge of autonomous hacking agents, incorporates special commands for handling interactive terminal functions, further increasing its potency but still requiring human intervention for some functions. These previous attempts reinforce the growing need to utilize LLMs in cybersecurity but also point out the need for fully autonomous agents capable of addressing complex, real-world scenarios independently. Our contribution aims to overcome these constraints by presenting CyberGuardian, an LLM-driven agent that can solve Capture The Flag (CTF) problems on its own, thus promoting autonomous penetration testing.

Back Ground

Capture The Flag (CTF) Challenges

CTF competitions assess competitors' abilities to identify and take advantage of weaknesses in virtual settings. Web exploitation, cryptography, reverse engineering, forensics,

binary exploitation, and general skills are just a few of the cybersecurity domains that present challenges. Some well-known CTF platforms are HackTheBox, TryHackMe, and Root Me, and some well-known competitions are DEF CON CTF, which has participants from around the world.
CTF challenges are cybersecurity exercises that task the players with finding security vulnerabilities in a test IT system. CTF challenges aim to find a text string called the "flag" hidden in purposely vulnerable software or a website. Web exploitation, cryptography, reverse engineering, forensics, binary exploitation, and general skills are just a few of the cybersecurity topics covered by CTF challenges.

Heuristic CTF Solvers
Traditional CTF solvers rely on heuristic-based software that automates some of the steps but is not flexible. Among the best of these are Katana and Remenissions that use predefined rules and tools to solve challenges but cannot match the creativity of the human expert.

Standard techniques for solving CTF challenges are generally founded on heuristic tools that perform a certain task but cannot be as adaptable as human minds. Katana is an open-source, general-purpose CTF solving environment based on brute-force attacks, employing a collection of pre-programmed tools to attempt to solve problems in numerous categories. Remenissions is a tool developed to solve binary exploitation challenges and decompiles the binary and searches for known vulnerabilities. While such tools can accelerate the process of solving the CTF, they cannot tailor themselves to unusual challenges nor develop new solutions.

LLMs in Cybersecurity
LLMs have been promising in several cybersecurity applications, from secure coding, vulnerability detection, malware identification, and test case generation. They can also be employed offensively, for example, to create malware or customized phishing emails.

LLMs find numerous applications in the domain of cybersecurity. There have been some interesting findings on the defensive side such as in secure coding, with evidence proving that codes written by humans with the assistance of LLMs are less buggy. It has been proven that LLMs are more effective in test case generation than prior methods. LLMs were found to be more effective in the identification of vulnerable code than static code analyzers. LLMs can assist humans in the malware detection process, but cannot replace them for now. It was also shown that LLMs can do automated fixing of vulnerable/buggy code. Additionally, LLMs can be used on the offensive side too, i.e., for hardware-level attacks, e.g., usage of side-channel analysis. LLMs can be utilized for software-level attacks, such as generating malware, and network-level attacks by generating personalized phishing emails.

LLM Agents
LLM agents are autonomous systems that can perceive, make decisions, and act. Symbolized by OpenHands, AutoDev, and Devika, they demonstrate coding and software engineering activities. LLM agents for CTF challenges, such as PentestGPT and HackingBuddyGPT, are promising but require human intervention for certain tasks.

LLM agents are autonomous systems based on Large Language Models that perceive their environment, choose proper actions, and execute actions as a result. LLM agents have been tried and tested thoroughly in a wide variety of applications from personal agents to agents performing machine learning experiments or trying to mimic human behavior. Many LLM agents have been implemented in software engineering. OpenHands is an open, standalone coding agent with over 30,000 GitHub stars which can code and solve programming assignments. Openhands has inspired a variety of similar general-purpose LLM-based agents like AutoDev, Devon, and Plandex. Devika is a computer science agentic AI who is able to interpret human instructions, break them up into commands, research, and code to accomplish a particular mission. SWE-agent is a commercial agent interface for computers that addresses the limitations of previous agents, such as having no access to an interactive terminal.

Datasets
There are various datasets which are particularly created to evaluate penetration testing agents and are usually compilations of Capture The Flag (CTF) challenges from competition or CTF websites. NYU CTF Benchmark is a collection of 200 CSAW CTF competition challenges (2017–2023) that are exemplary security problems in six categories of varied difficulty levels. Intercode CTF Benchmark includes 100 PicoCTF challenges in six categories. But it lacks difficulty levels and dynamic flags because the flags and files are statically stored, limiting its flexibility. The Cybench Framework offers 40 professional-level CTF challenges from four competitions, each divided into subtasks for complete analysis. It is modular and offers tasks that are designed to mimic real-world hacking capabilities, such as exploiting vulnerabilities. Additionally, problem sets on sites like Hack The Box have been utilized to challenge agents, although no comprehensive benchmark has as yet been established for them. Both together provide a diverse collection of tools for measuring the proficiency of cybersecurity agents.

III. PROPOSED METHODOLOGY

The approach to CyberGuardian described here is a two-module system of a Planner and a Summarizer with the objective of iteratively generating and executing commands for autonomous penetration testing. The Planner takes as input an LLM that derives executable commands based on the system state and the condensed output of the Summarizer,

which contains a complete condensation of all actions and observations. This circular interaction is repeated until the task is completed or an iteration count set in advance is reached. For security, CyberGuardian executes within a container environment with a firewall, restricting network access and inhibiting unauthorized interactions. For testing, two new benchmarks based on PicoCTF and OverTheWire were designed, with 200 challenges in various domains and levels of difficulty. These standards include challenge descriptions, tips, file paths, categories, difficulties, and dynamic solver functions to ensure solid and robust testing. Comprehensive testing of Cyber Guardian's functionality and the potential of LLM-based agents in self-governing cybersecurity solutions is made easier by this robust platform.

V. METHODS

This study makes use of Cyber Guardian, a dual-module design that automatically addresses cybersecurity vulnerabilities using Large Language Models (LLMs). The framework consists of a Summarizer module that keeps a brief record of activities and observations to direct further steps and a Planner module that creates commands to promote task completion. To reduce the dangers connected with command execution, both modules function in a safe, containerized environment. Additionally, Cyber Guardian's performance is assessed using two extensive benchmarks based on the PicoCTF and OverTheWire platforms, which span a variety of challenge categories and difficulty levels. This methodology facilitates a thorough evaluation of the framework's aptitude for resolving intricate cybersecurity issues.

1. CyberGuardian Architecture

CyberGuardian consists of two primary modules: the Planner and the Summarizer. The Planner generates commands to be executed in a containerized Kali Linux environment, while the Summarizer maintains a comprehensive summary of all actions and observations. This dual-module architecture enables CyberGuardian to iteratively execute commands and adapt its strategies based on feedback.

2. Planner Module

The Planner generates executable commands from the current system state and summarized outputs from the Summarizer. It uses an LLM to interpret the context and generate the best fit command, surrounded by <CMD></CMD> tags.

The Planner module generates actionable commands that bring the system closer to finishing tasks. The planner uses an LLM to draw conclusions about the system's current condition and the condensed results of previous instructions that the Summarizer module has issued. Based on these inferences, the Planner constructs new commands with the aim of making progress. The system prompt to the Planner is formulated to yield one, terminal-executable command that meaningfully makes progress toward the task. The directive instructs the LLM to act like a seasoned penetration tester that is solving a Capture the Flag (CTF) challenge. Indicating CTF context is necessary to prevent rejection of prompts by the LLM due to ethical concerns. Special directions are employed to ensure the LLM does not reproduce commands, utilizes the full system state present, and is extra cautious while coming up with the most appropriate command at every step.

3. Summarizer Module

The Summarizer updates the history of action and results so that the system maintains an explicit record of progress. It sanitizes and summarizes command outputs to provide concise summaries, necessary to maintain context without overwhelming the Planner.

The Summarizer module compensates the Planner in that it continuously keeps updating the action history and the results so that the system acquires a concrete history of how much it has already gone. The Summarizer is also LLM-based, and its function is to summarize and polish the output of each command drafted by the Planner. It shows a running summary by adding new information on every completed command. Also, this module is important because most of the commands produce lengthy outputs with occasionally minimal useful information, therefore, this module is also an essential filtering module. The Summarizer module is at the heart of context preservation, as it allows the system to understand what has been done, what has been generated, and how to use that to inform the next step without employing too big context windows by integrating all previous commands and their respective outputs into the Planner prompt.

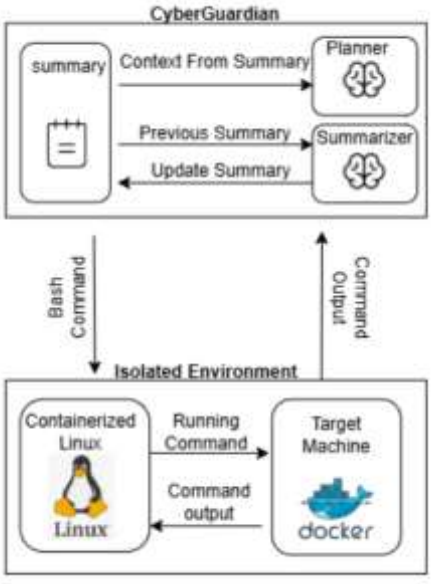

Fig 1 : Architecture of cyberGuardian

4. Operational Workflow

The operational workflow of CyberGuardian involves a cyclical interaction between the Planner and Summarizer modules within the constrained execution environment:

- **Command Generation:** The Planner generates a command based on the current summarized history, aiming to progress toward capturing the flag.
- **Command Execution:** The generated command is executed within the containerized Kali Linux environment.
- **Output Summarization:** The output from the command execution is forwarded to the Summarizer, which updates the summarized history.
- **Iteration:** The updated summary is returned to the Planner for the next command generation cycle. This cycle keeps on until either the flag is caught or the maximum number of iterations is achieved. By systematically utilizing the strengths of LLMs in planning and summarization, CyberGuardian effectively solves complex cybersecurity challenges.

5. Securing Cyber Guardian

There are significant security ramifications in implementing Cyber Guardian as a standalone LLM-driven agent that can run terminal commands. The most significant concern is that the agent misunderstands its missions and starts out-of-scope engagements with the target in an unauthorized manner. Additionally, running commands on the host system ups the ante of the agent performing malicious actions locally.

Cyber Guardian runs in a containerized environment to reduce such threats. By compartmentalizing the agent from the host system, this environment blocks the execution of commands from causing unforeseen effects, including devastating file operations like rm -rf. To keep the agent from reaching out to hosts that are beyond its scope, a firewall is set up to limit the network reachability of the intended target computer only.

But challenges exist in deploying an effective firewall. It is unsafe to create rules for the firewall within the container space because the agent may overwrite them if it achieves too much access. As an alternative, rule definition outside of the container makes deployment on multiple systems cumbersome and hinders generalizability.

Our remedy is to overwrite firewall rules prior to running any commands that Cyber Guardian generates. But the agent can still evade such safeguards—i.e., by utilizing a cron job to modify the firewall settings and target devices beyond its reach. Also, the agent can evade our limitation by targeting its target via the target machine in case it has access to the internet via some network to which it is connected.

IV. EXPERIMENTAL RESULTS

This section contains the experimental findings pertaining to Cyber Guardian's performance assessment and parameter optimization on the two suggested Capture the Flag (CTF) benchmarks, PicoCTF and OverTheWire. The experiments are organized into two distinct phases. The first phase explores the optimization of key parameters using the Microsoft Phi-3-mini-4k model, while the second phase assesses its performance on the benchmarks with the optimized settings. The benchmarks encompass a variety of cybersecurity domains, including General Skills, Web Exploitation, Cryptography, Binary Exploitation, Forensics, and Reverse Engineering, with challenges classified by difficulty levels: Easy, Medium, and Hard.

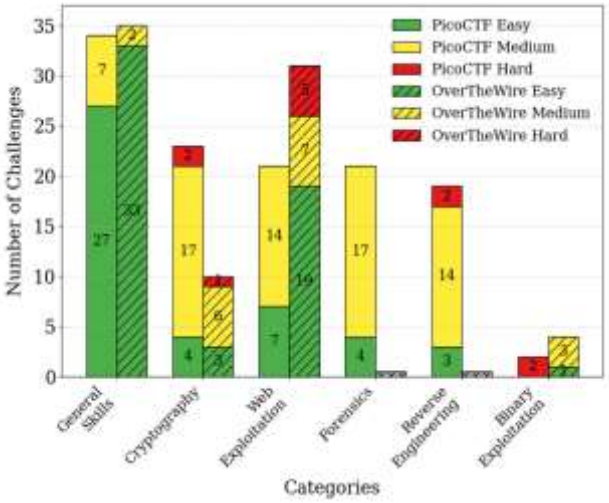

Figure 1 illustrates the distribution of benchmark challenges across various categories and difficulty levels for two platforms: (a) PicoCTF and (b) OverTheWire. The number of problems in each category—General Skills, Web Exploitation, Cryptography, Binary Exploitation, Forensics, and Reverse Engineering—is graphically represented by the bar charts. These difficulties are further divided into three levels based on their level of difficulty: Easy, Medium, and Hard.

1. Parameter Tuning Exploration

Optimizing the parameters of the Phi-3-mini-4k model within Cyber Guardian is essential to enhance its effectiveness on the CTF benchmarks.The parameter tuning focused on three primary parameters: the temperature, the top-p (nucleus) sampling parameter, and the size of the observation window, which determines the number of maximum characters retained from every command output for summarization.

The window size has a significant effect on performance. Experiments revealed that increasing the window size from 0 to 250 characters improved the number of correctly solved challenges for the PicoCTF benchmark, as short windows did

not capture enough context to make well-informed decisions. Beyond 250 characters, however, performance did suffer due to including too much irrelevant, redundant information to confuse the summarizer. For PicoCTF, a window of 250 was found to be the optimal compromise, providing just enough information without being overwhelming. For the OverTheWire benchmark, benefits of a larger window were less pronounced, at least in significant part because commands like curl and ssh created redundant boilerplate text as leads for output, necessitating the larger window—at 500 characters—to be able to encompass critical information. Nevertheless, very large windows risked destroying emphasis on critical information. These results indicate that the optimal window size depends on the benchmark, with PicoCTF preferring moderation and OverTheWire needing more space to accommodate verbose output.

The temperature parameter of variability of token selection was also experimented with. Low values (e.g., between 0 and 1) created more systematic and predictable outputs appropriate for generating dependable commands, and larger values boosted diversity and therefore could be helpful in exploratory problem-solving. Until a temperature of 1, performance was stable, it falling below this due to less coherent commands occasionally disturbing the system environment (e.g., by altering configurations unmeaningfully). Error rates also followed this trend, being low up to a temperature of 1 and then rising. The temperature of 1 was selected because it balanced reliability against the right degree of flexibility for CTF tasks.

The top-p hyperparameter, controlling the diversity of tokens to be sampled, was tuned across a range from 0.1 to 1.0. Small values restricted the model to strongly confident tokens with precision assured, while higher values expanded the token set, forcing diversity. Slight improvement in challenge completions was achieved when top-p reached 0.9, while moderate improvements resulted from the ability of the model to sample more diverse sets of commands. This context also increased the use of lower-frequency commands in PicoCTF, but OverTheWire challenges used frequently used commands like ssh and curl very effectively, which was not very variable. The top-p value was used as 0.9 in an effort to optimize performance at the cost of coherence.

Other setups were attempted, including sampling, which enhanced performance by 38% by introducing randomness, and prompt chaining, which reduced performance by 5% and added 17% to processing time due to excessive token processing. All of these contributed to the final configuration with sampling on and prompt chaining off.

2. Performance Evaluation on Benchmarks

With the parameters that were optimized—temperature of 1, top-p of 0.9, sizes of observation windows of 250 for PicoCTF and 500 for OverTheWire, and 20 maximum iterative steps—Cyber Guardian's performance was measured with both benchmarks. Twenty planning and summarizing iterations were allowed in each challenge in order to proceed towards flag capture, and there was no pruning.

On the PicoCTF test, the system was good in many categories with the 250-character observation window providing sufficient context for effective summarization. Progress was smooth, with completion accumulating at a faster rate earlier in the steps as easier problems were completed and then slowing towards the end as complexity increased. Temperature and top-p settings provided a balance between stable and exploratory commands to make steady progress without catastrophic error.

For the OverTheWire benchmark, the larger window size of 500 characters accommodated the verbose outputs typical of its challenges, particularly those requiring extended interaction via ssh or curl. However, the redundant nature of such outputs sometimes obscured critical details, making progress more cumbersome compared to PicoCTF. Completion times per challenge were quicker on OverTheWire, likely due to differences in summary length depending on task organization and prompt formulation of the benchmark.

The iterative experiment revealed that incremental steps added progressively increasing completions, though the increment diminished as the system exhausted simpler challenges. Computational cost increased linearly with steps, fueled by mounting input summaries, with command outputs remaining short. Token utilization plateaued after around 10 steps, which revealed a bound in summary size despite ongoing information accumulation. Altering the size of observation windows also did not have a significant impact on total token use, since reductions from 500 to 100 characters reduced total use by under 5%, since the summaries tended towards an average size irrespective of input size.

In general, the Phi-3-mini-4k configuration, with the optimized parameters presented here, produced stable performance across both benchmarks. PicoCTF benefited from the balance between context windows and command diversity, while OverTheWire required tuning for its lengthy style. These findings highlight the importance of fine-tuning parameters to specific benchmark characteristics so that Cyber Guardian is able to proficiently address comprehensive cybersecurity tasks.

V. CONCLUSION & FURTHER SCOPE

Major advances to autonomous penetration testing by a combination of Large Language Models (LLMs) based on a two-module architecture comprising a Planner and a

Summarizer are highlighted in this study. The platform possesses the ability to iteratively synthesize executable commands and process feedback, and it provides adaptive penetration testing in an unsupervised manner. Standardized Capture The Flag (CTF) testing through tools like PicoCTF and OverTheWire provides a good test bed to compare LLM-based systems against diverse cybersecurity tests. Experimental results show that LLMs are superior to conventional heuristic tools as far as flexibility and problem-solving are concerned, but security concerns like hallucinated IP addresses or improper action call for stringent security guards.

The horizon for the future is to expand autonomy, build multi-agent cooperation, and expand applications to encompass fields like threat intelligence and autonomous vulnerability analysis. Continuous learning mechanisms and simplicity are crucial to respond to changing threats and make it easier for cybersecurity professionals to access. Public release of benchmarks and code can assist in innovating autonomous cybersecurity solutions by providing a basis for more effective and efficient countermeasures against sophisticated cyberattacks.

REFERENCES

1. Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, “A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly,” High-Confidence Computing, p. 100211, 2024.
2. G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, “PentestGPT: An ŁŁM-empowered automatic penetration testing tool,” arXiv preprint arXiv:2308.06782, 2023.
3. J. Xu, J. W. Stokes, G. McDonald, X. Bai, D. Marshall, S. Wang, A. Swaminathan, and Z. Li, “AutoAttacker: A large language model guided system to implement automatic cyber-attacks,” 2024. [Online]. Available: https://arxiv.org/abs/240.01038
4. K. Leune and S. J. Petrilli Jr, “Using capture-the-flag to enhance the effectiveness of cybersecurity education,” in Proceedings of the 18th annual conference on information technology education, 2017, pp. 47–52.
5. Carnegie Mellon University. (2024) PicoCTF. Accessed: November 1, 2024. [Online]. Available: https://picoctf.org/
6. OverTheWire. (2024) OverThewire wargames. Accessed: November 1, 2024. [Online]. Available: https://overthewire.org/wargames/
7. “Hack The Box - Hacking Training Platform,” https://www.hacktheb ox.com, accessed: 2024-10-02
8. TryHackMe - Cyber Security Training Platform,” https://www.tryh ackme.com, accessed: 2024-10-02.
9. T. Balon and I. Baggili, “Cybercompetitions: A survey of competitions, tools, and systems to support cybersecurity education,” Education and Information Technologies, vol. 28, no. 9, pp. 11 759– 11 791, 2023.
10. F. N. Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, “Large language models in cybersecurity: State-of-the art,” arXiv preprint arXiv:2402.00891, 2024
11. F. N. Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, “Large language models in cybersecurity: State-of-the art,” arXiv preprint arXiv:2402.00891, 2024
12. Y. Li, H. Wen, W. Wang, X. Li, Y. Yuan, G. Liu, J. Liu, W. Xu, X. Wang, Y. Sun, R. Kong, Y. Wang, H. Geng, J. Luan, X. Jin, Z. Ye, G. Xiong, F. Zhang, X. Li, M. Xu, Z. Li, P. Li, Y. Liu, Y.-Q. Zhang, and Y. Liu, “Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security,” 2024. [Online]. Available: https://arxiv.org/abs/2401.05459
13. X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig, “OpenHands: An open platform for ai software developers as generalist agents,” 2024. [Online]. Available: https://arxiv.org/abs/2407.16741.
14. M. Shao, S. Jancheska, M. Udeshi, B. Dolan-Gavitt, H. Xi, K. Milner, B. Chen, M. Yin, S. Garg, P. Krishnamurthy et al., “NYU CTF Dataset: A scalable open-source benchmark dataset for evaluating llms in offensive security,” arXiv preprint arXiv:2406.05590, 2024.
15. W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, “A Survey of Large Language Models,” 2024.