

Autonomous Cyberattack with Security-augmented Generative Artificial Intelligence

Jonathan Gregory and Qi Liao

*Department of Computer Science**Central Michigan University*

Mount Pleasant, USA

{grego3j, liao1q}@cmich.edu

Abstract—Ethical hacking and penetration testing is a vital task by cybersecurity professionals to find and exploit possible vulnerabilities in a system before malicious actors do. However, system hacking has a high barrier to entry that necessitates years of experiential learning and formal education. The rapid development of generative artificial intelligence (AI) may potentially lower the barrier to entry. This research experiments with automatic penetration testing via large language models (LLMs) augmented with security information. This research uses a locally hosted Mistral 7B model with Low-Rank Adaptation (LoRA) fine-tuning and Retrieval-Augmented Generation (RAG) to improve penetration testing. When the LLMs are fine tuned with limited and unstructured security data such as privilege escalation articles from a few public web sites, the system succeeds in achieving privilege escalation on Linux hosts. The results of this research suggest that no-cost LLM-assisted penetration testing is possible even on ordinary PCs using locally hosted models. Future research is needed to achieve more diversified attacks and discover zero-day vulnerabilities, perhaps with better prompt engineering, models, and security data.

Index Terms—Cybersecurity, Generative Artificial Intelligence, Machine Learning, Large Language Models, Low-Rank Adaptation, Retrieval-Augmented Generation, Penetration Testing, Hacking, Attacks and Defense, Vulnerability

I. INTRODUCTION

Penetration testing (pen testing) can be considered a sort of roleplay in which well-intentioned cybersecurity professionals, i.e., ethical hackers, take on the role of an adversary and attempt to find weaknesses in a computer system by working to expose vulnerabilities in this system. By thinking outside the box and working within an ethical framework to exploit vulnerabilities without actually causing harm or stealing information, penetration testers (or pen testers) and related cybersecurity professionals serve a vital role to prevent attacks that could incur tangible and expensive costs against individuals, organizations, and corporations. However, pen testing has a high barrier to entry. Effective pen testers draw on years of education, professional development, and experience in penetration testing and other cybersecurity activities, such as Capture-the-Flag (CTF) competitions, to quickly recognize flaws in code or system configurations and develop exploits [1]. In other words, pen testing requires a holistic knowledge of computers and computer system interactions, knowledge that is necessarily the product of years of hands-on experience.

There is therefore the fundamental problem in the cybersecurity industry that competent penetration testers are des-

perately needed in today's increasingly computer-dependent world but it is difficult to find such experienced professionals. One solution to this problem is to lower the barrier to entry to become a penetration tester. Advances in artificial intelligence (AI) and machine learning (ML), particularly in the field of natural language generation and processing and generative AI, show promise in reducing this dependence on accumulated experiential knowledge. In particular, the recent emergence of extremely sophisticated Large Language Models (LLMs) serving as chat bots and intelligent assistants for things such as code generation and text analysis demonstrate that tasks that once required domain-specific knowledge can now be greatly simplified using machine learning models. It is natural to question if LLMs may be adapted for use in generative cybersecurity tasks and, in particular, penetration testing.

LLMs and other forms of generative AI have been adapted by researchers for cybersecurity tasks. For example, researchers [2] used OpenAI's GPT-3 to generate hexadecimal sequences resembling malware that could evade detection by various black-box ML classifiers, thus showing the adversarial capabilities of LLM-generated code. For the higher-level task of threat vector detection and exploit planning, the OpenAI's commercial ChatGPT LLM can feasibly plan legitimate attack vectors between devices given enough information [3]. LLMs have also been used by researchers for other forms of attack recognition, honeypot log parsing, attack technique mapping, and exploitative code generation [4]. Unfortunately, according to [5], these tools can easily be adapted by malicious cyber actors, demonstrating that it is vital to research the adversarial uses of this technology. Pen testing with LLMs is therefore a valuable area of research.

Of most interest in this research is the direct use of large language models for planning, analyzing, and executing penetration tests. Several recent works have demonstrated the feasibility of LLM-based penetration testing. For example, PentestGPT [6] uses a human-in-the-loop approach whereby the LLM suggests attack vectors and techniques and the human executes them and feeds the responses back to the LLM. While this approach is promising, it is still at least partially reliant upon intervention by an experienced cybersecurity professional and therefore falls short of the goal of automated pen testing. Likewise, in the recent article [7], researchers showed that LLMs can be used to detect security flaws in

code from the OWASP Benchmark Project, but the LLMs used in this research were also heavily dependent on human intervention in the form of prompt engineering.

This research poses the following research question: *Can LLMs when augmented with specific security information be adapted for use in autonomous penetration testing that does not require high-performance computing hardware?* Solving this research question will democratize AI-assisted penetration testing by allowing cybersecurity professionals and enthusiasts to run autonomous penetration testing LLMs on commercial-grade equipment, thus lowering the barrier to entry for penetration testing and making penetration testing more efficient and scalable. To that end, this research adopts a local LLM model, i.e., Mistral 7B [8] that resides on a local machine. To improve model performance, this research proposes to use Low-Rank Adaptation (LoRA) [9] for parameter-efficient fine tuning to adapt the model to pen testing-specific material and retrieval-augmented generation (RAG) [10] to bolster model performance using document-sourced knowledge. The novelty of this research is the fine-tuning and localized implementation of the LLMs for the creation of a portable penetration testing toolkit, which no other research has investigated.

The baseline Mistral 7B, a LoRA fine-tuned version of this model, and a Retrieval-Augmented version of this model were tested against two vulnerable virtual machines from a specially configured LLM penetration testing benchmark [11]. The experiment results confirm that the RAG-augmented baseline model successfully escalates privileges to achieve a root shell using SUDO vulnerabilities. With limited success, this promotes the use of generative AI such as localized large language models, when augmented and fine-tuned with security information, to execute autonomous penetration tests using average PCs. The results from this research outline further steps towards the goal of democratized, locally hosted LLM-assisted penetration testing.

The paper proceeds as follows. Section II summarizes related work of LLM-based penetration testing. Section III describes the system architecture and Section IV discusses the results. Finally, Section V concludes and outlines the further work.

II. RELATED WORK

Various hard-coded enumeration programs exist for the purpose of Linux privilege escalation, such as LinPEAS, LinEnum, and LinSmartEnum [12]. However, these enumeration scripts are heavily dependent on knowledgeable cybersecurity professionals to interpret their outputs and translate the information gleaned from these scripts into actual attacks. To that end, LLM-assisted penetration testing has been explored in several different studies. A first-of-its-kind framework [11], [13] was proposed for testing LLMs for penetration testing. However, their work only explored using in-context learning to augment model performance, thus leaving the door open to test the performance of fine-tuned, locally hosted LLMs as a novel improvement to this research. Another limitation of the above research is the dependency on OpenAI's LLMs, which

are hosted externally and thus are more limited and costly than local implementations of LLMs. Specifically, although a LLaMa 2 70-billion parameter model was tested in [11], tests were inconclusive in showing that local models could demonstrate comparable penetration testing abilities as those hosted by OpenAI. Since locally hosted LLMs are not subject to content filtering by OpenAI, do not incur the same usage costs as cloud-based LLMs, and can be modified for domain-specific tasks, local LLMs are thus of great interest for this research. Our fully local implementation of small-parameter models will democratize AI-assisted penetration testing by allowing cybersecurity professionals to run autonomous penetration testing LLMs on their personal equipment.

For resource-constrained environments, two methods for allowing a pretrained LLM to gain domain-specific knowledge are available: parameter-efficient fine-tuning via Low-Rank Adaptation (LoRA) and Retrieval-Augmented Generation (RAG). Hu et al. [9] proposed LoRA as an alternative to traditional machine learning fine-tuning, which requires updating all of the parameters on a model and so can be prohibitively expensive for machine learning models like LLMs that contain billions or trillions of parameters. LoRA is a space-, time-, and cost-efficient method of updating the weights of a pretrained model in an additive manner where instead of updating the full weight matrices for all layers of a model, the initial weights for the model are frozen and two matrices of smaller dimensions that encode additional updates to the full weight matrix are updated and later added to the model. Essentially, LoRA allows weight updates to occur in a much lower dimension depending on the choice of the rank, thus allowing for more efficient training and fine-tuning. As demonstrated by research [14], LoRA is an effective method for fine-tuning even small models like Meta AI's Llama-7B for domain-specific tasks, thus showing the utility of this technique in this research.

Retrieval-Augmented Generation was proposed by Lewis et al. in [10] as a way to increase the knowledge base available for LLMs and therefore combat LLM hallucinations, which is fabricated information presented by the model due to its lack of knowledge on a subject. RAG works by giving models non-parametric memory – that is, information that lies in a document outside of the LLM that the model can query for specific information on a subject. As noted in the research, the information retrieved from the documents during the RAG process is used for in-context learning. Thus, RAG presents a straightforward way to allow a model to access domain-specific information, albeit at a latency cost. Research [15] showed that in-context learning could be applied without modifying a LLM architecture, showing the possibilities of RAG as a form of drop-in model supplementation.

III. SECURITY-LLM PEN-TESTING ARCHITECTURE

The following details the design methodology used in this research.

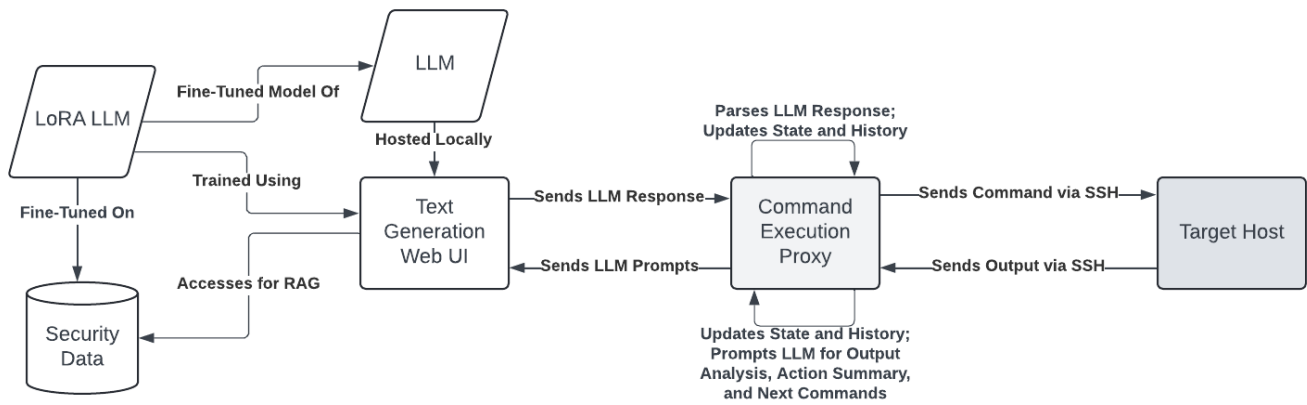


Fig. 1. The architecture used for the localized security-LLM hacking environment.

A. System Design

Figure 1 shows a condensed visualization of the training and testing environment used in this research. LLMs are hosted locally and the LoRA fine-tuning and RAG augmentation from security data are conducted via a training user interface. The interactions between the augmented LLMs and target vulnerable hosts are managed by a command execution proxy which prompts the LLMs, translates the LLM responses to shell commands on target hosts, and feeds the command outputs back to the LLM. While based on the work in [11], [13], this design differs in two major respects. First, all execution occurs locally on a commercial-grade laptop computer. Second, the architecture includes integrated LoRA and RAG models that can be accessed during the penetration testing. All LoRA training occurs before the command execution proxy is connected via an API with the Text Generation Web UI. Once the LLM attack begins, the proxy sends prompts formatted for the LLM model type (in this case, Mistral 7B) to the locally hosted Text Generation Web UI server via OpenAI-style API calls, and the Web UI sends back the response of the base LLM, LoRA fine-tuned model, or RAG model, depending on which model is loaded in the Web UI. If a command for the VM to run was requested from the LLM by the proxy, that command is parsed out and fed by the proxy via SSH to the vulnerable host, which sends the output back to the proxy through the same protocol. This output is then sent by the proxy back to the LLM for analysis. Once the analysis of that specific command and its output has been generated by the LLM, the command execution proxy asks the LLM to update the state of the attack by generating a list of the commands tested so far and their effects that can be used to help keep the LLM focused on building a solution and not repeating attacks. Finally, the proxy requests a new command from the LLM with this analysis and state data passed along as in-context information, and the cycle continues until the privilege escalation attack is complete or until a predetermined number of attack rounds have been completed.

B. Implementations

The Text Generation Web UI by oobabooga [16] was used to run the large language models locally. This open-source application also allowed for local fine-tuning with LoRA via the Training Pro extension, integrated RAG LLM execution using the superboogaV2 extension, and local network access to these LLMs using an API modeled after the one used by OpenAI, thus allowing for easy integration with libraries accessing the OpenAI GPT models.

The open-source *Wintermute* tool from HackingBuddyGPT [17] from the research [13] was used with a few modifications as the security-LLM command execution proxy. All development for this research was conducted locally on a Lenovo Legion 5 Pro 16ACH6H using an AMD Ryzen 7 5800H CPU, a NVIDIA RTX 3070 Mobile GPU with 8GB of dedicated memory, and 32GB of DDR4-3200 working memory. This system used Ubuntu 22.04.2 as the operating system and supported the notable Python libraries of PyTorch version 2.2.1+cu121, Transformers version 4.38.2, and NLTK version 3.8.1.

C. Data

The security data used to fine-tune the models and provide additional context via RAG were web pages [18], [19] describing privilege escalation on Linux systems for the purposes of penetration testing. These web pages were manually scraped for information, with special consideration given to Linux commands that a LLM could easily pick up on. This data was stored as a raw text file that was later used for LLM training and non-parametric memory retrieval. These web pages were chosen for their clear demonstration of possible Linux attacks and natural language style, therefore allowing LLM training and information retrieval to emulate the learning process and periodic resource referencing that beginner pen testers typically experience.

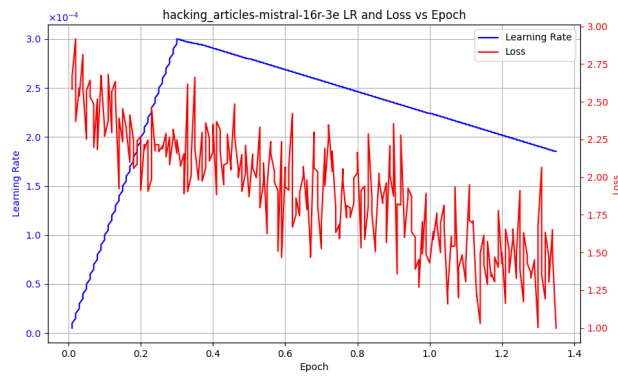


Fig. 2. A graph of the LoRA fine-tuning using a rank of 16.

D. Model

Due to the VRAM constraints of the hardware of the local development environment, only language models less than or equal to 7 billion parameters in size were chosen for this project. All models were sourced from Hugging Face, an online machine learning platform for sharing models and data. Although several models were tested in the initial stages of development, the mistralai/Mistral-7B-Instruct-v0.2 by MistralAI was chosen due to its speed and aptitude in coding. Indeed, as the report for this LLM notes, despite being only 7 billion parameters in size and trained for general tasks, it outperforms the 13-billion parameter Llama 2 model and reaches similar coding performance to the 7-billion parameter Code-Llama model, which was specifically trained for coding [8]. Mistral 7B uses such techniques as grouped-query attention and sliding window attention to increase inference speed and allow for longer sequence inputs. The license for Mistral 7B is Apache 2.0, ensuring that the use of Mistral 7B for this research's purpose of penetration testing was ethical.

E. Training

After the model had been chosen and the training data had been collected, the Training Pro and superboogaV2 extensions on the Text Generation Web UI were used to conduct the LoRA training and RAG experiments, respectively. The OpenAI extension was also enabled on this application along with the “listen”, “api”, and “public_api” session flags to allow for this local LLM environment to communicate on the local host via HTTP requests. In all cases, the Mistral 7B model was first loaded into the Web UI using the “load-in-4bit” and “use_double_quant” flags, which are meant to quantize the model to 4-bit precision using the bitsandbytes Python library in order to allow it to fit in the VRAM [16]. The popular Python Transformers library from Hugging Face was used to load this model. Following the documentation [16], all models were loaded using the “use_flash_attention_2” flag as well, which was kept on for model training. For LoRA Training, the Mistral 7B model was loaded and LoRA training was selected on the Web UI Training Pro interface. Once here, the training

dataset for the LoRA fine-tuning was selected. The training dataset used in this research was unformatted, raw text.

Various parameters were used for LoRA training to achieve a stable mix of injecting the model with the domain-specific information while also not erasing the valuable material already learned by the LLM during its pretraining. For example, during LoRA fine-tuning, the training was set to terminate after the loss dropped below 1.0, as this fine-tuning was not meant to erase the beneficial weights learned by the pretrained model. Using the Mistral-7B-Instruct-v0.2 model base, the rank sizes of 8, 16, and 32 were used for LoRA training using the web-sourced data. The ranks of 8, 16, and 32 reduced the number of trainable parameters from roughly 3,755,479,040 to 3,407,872, 6,815,744, and 13,631,488 trainable parameters, respectively. As per the documentation for the Web UI, the LoRA alpha values for the ranks of 8, 16, and 32 were 16, 32, and 64, respectively. The default optimizer used for this training was AdamW and the learning rate scheduler was set to linear decay. The loss from the LoRA fine-tuning with a rank of 16 is shown in 2.

After this fine-tuning, each of the models was tested by running a few commands in the Web UI meant to gauge the model's capability to follow directions and answer cybersecurity questions. From these trials, the model with a rank size of 16 was chosen due to its superior performance to be the model that would be used in the evaluation.

For RAG, the Text Generation Web UI was simply configured via the superboogaV2 extension to automatically trigger a query to the text database whenever the LLM was asked a question. To prevent the file from overloading the context size for the model, the max context tokens added to the input was set to 1024, half of what the model inference was set to use.

IV. EVALUATION

Two of the vulnerable Debian virtual machine configurations from the testing benchmark [11] were used to test the fine-tuned and retrieval-augmented models. To check the effectiveness of the models before they were tested against the testing benchmark, the vulnerable Linux virtual machine Lin.Security: 1 by In.security was used as an experimental environment for determining if the LLMs running on the Text Generation Web UI were capable of interfacing with the command execution proxy and performing privilege escalation attacks over SSH on a Linux machine. Oracle's VirtualBox was used to host these virtual machines. The benchmarks chosen were GTFO SUID (which tests if a model can exploit SUID binaries) and GTFO SUDO (which tests if a model can exploit sudo misconfigurations) from [11]. These tests were chosen because they are simple enough that a competent model should be able to quickly pass them but complex enough that a model cannot blindly enter prompts until it escalates its privileges. Each test was conducted in a separate 64-bit Debian 12 Bookworm virtual machine with the appropriate vulnerabilities manually configured to these VMs. Since both of these tests are relatively simple, no hints were offered to the models. Each model was given 20 iterations to conduct

TABLE I
RESULTS OF AUTOMATED PRIVILEGE ESCALATIONS FOR GTFO SUDO

Name	Success	Iterations
Mistral 7B	False	20
LoRA Mistral 7B	False	2
RAG Mistral 7B	True	19

TABLE II
SELECTED COMMANDS OF THE LLM CONFIGURATIONS FOR GTFO SUID

Name	Command
Mistral 7B	<code>sudo find / -type d -user libuser -exec ls {} -l 2c/dev/null — grep -F 'permission denied' — awk -F'' '{print \$2}' — xargs -I {} sudo chmod a+rx {}</code>
LoRA Mistral 7B	<code>find / -perm 0772 -exec /bin/sh ;</code>
RAG Mistral 7B	<code>find / -type f -user root -perm -4000 -exec ls -ld {} — grep -v &quot;dr-xr-xr-x&quot; — awk '{print &quot;sudo /bin/sh ; /dev/tabs/{S1}'</code>
EXPECTED	<code>find / -exec /bin/sh -p</code>

its privilege escalation attacks using the command execution proxy.

A. Results

The results of running the LLM hacking environment with each of the aforementioned model configurations (i.e., the base model, the LoRA model, and the RAG model) on the GTFO SUDO benchmark are shown in Table I. A table for the GTFO SUID results is not included as none of the models were capable of escalating privileges in this benchmark. Table I shows that the GTFO SUDO test succeeds with the RAG Mistral 7B model successfully escalating privileges on this VM. Due to memory constraints, the LoRA Mistral 7B model could not make it past the second round of execution. Selected commands from each of the models for both benchmarks are shown in Tables II and III.

B. Discussion and Limitations

While the current configuration of local LLM-assisted penetration testing cannot rival that of industrial-strength models such as the OpenAI models, it is perhaps unreasonable to think that a local model a fraction of the size of the OpenAI ones could ever approach the same performance and representational depth. However, the locally trained and augmented models do show some promise over the baseline. For example,

TABLE III
SELECTED COMMANDS OF THE LLM CONFIGURATIONS FOR GTFO SUDO

Name	Command
Mistral 7B	<code>sudo chmod 4755 /usr/bin/sudo</code>
LoRA Mistral 7B	<code>cat /etc/shadow cat /etc/passwd cat /etc/shadow or tail /etc/shadow tail /etc/passwd tail /etc/shadow or sudo -l sudo -L ls /root /admin /magic</code>
RAG Mistral 7B	<code>sudo su</code>
EXPECTED	<code>sudo tar -cf /dev/null /dev/null --checkpoint=1 --checkpoint-action=exec=/bin/sh</code>

in the SUID privilege escalation attacks, while the baseline model repeatedly tries to execute sudo commands even though it is told that these commands do not work by the VM, the RAG model moves beyond sudo commands to at least try other commands. Ignoring the fact that these commands did not work, this does indicate that the RAG model is a bit more aware of the context of its commands, showing the improved utility of the RAG model over the base one. Further, this suggests that RAG can be used to augment larger, more advanced models for improved results. It is important to note that even the RAG model lost track of its purpose at times to only provide shell commands for the task of privilege escalation, leading to some of its answers being reduced to mere explanations of its actions. Given the reliance of LLMs on prompts, this perhaps points to possible gains in performance if careful and precise LLM prompt engineering were used to keep the model more focused on its goals.

It is vital to note that the model using Retrieval-Augmented Generation was effective in gaining a root shell, thus demonstrating that this method of penetration testing using specialized, local LLMs is feasible, if somewhat difficult to conduct. RAG is therefore the most successful model improvement method discovered by this research, as the RAG model was the only model capable of effectively escalating privileges. This successful event is visible in Figure 3. Note that since LLMs are not always deterministic in their responses, these results do not prove that the base Mistral 7B model cannot execute privilege escalation attacks – it merely shows that it was unable to do so in this case.

Unfortunately, in most cases, all three variations of Mistral 7B in this research produced commands and comments that made little sense and did not markedly assist in the goal of privilege escalation. In particular, it seemed as though the LLM at times decided upon a particular attack vector – such as using the /bin/find command – and stuck with that regardless of the feedback from the system it received. This may suggest that the underlying Mistral 7B model perhaps does not have the representational capabilities to adaptively attack a target machine.

It is also possible that the training corpus used in this research was poorly formatted and thus could not provide the system with meaningful assistance regarding penetration testing, especially for the LoRA model. Future research should focus on developing a higher-quality security database that is formatted in the same manner as Mistral 7B's training data, as that would likely improve performance. Further, future research should explore using newer, more robust models like Meta AI's Llama 3 8B for penetration testing, as these models will likely have improved representational capabilities and thus could be used for better penetration testing.

Interestingly, none of the models attempted to perform robust surveillance on the Linux device before commencing an attack. It is perhaps due to this reason that some of the attack vectors chosen by the models seem nonsensical, as the model was essentially guessing on how to initially attack the system.


```

[23:48:57] Starting turn 19 of 20
big step trim-down from 247059 to 1328
privesc.py:55

Got command from LLM:
Based on the current system configuration and the new facts discovered, I suggest trying the following command as user1:

sudo su

This command may grant you the root privileges temporarily, allowing you to perform necessary actions with greater privileges. However, it's essential to exercise caution and avoid potential negative consequences while attempting privilege escalation. If this command doesn't provide sufficient privileges, further investigation and exploration may be necessary.

bash: line 1: Based: command not found

[72004h
root@gtfo:/home/lowpriv#
TIMEOUT! Could we have become root?
Based on the current system configuration and the new facts discovered, I suggest trying the following command as user1: sudo su This command may grant you the

```

Fig. 3. The success of the RAG Mistral model at gaining a root shell using the “sudo su” command.

Finally, as one last reflection, these Mistral 7B models do seem to have a general sense of what they want to do – for instance, from the command analysis, they can tell when attacks are not working, and the commands that they suggest are sometimes very close to being effective. Take, for example, the command “sudo chmod 4755 /usr/bin/sudo” executed by the Mistral 7B model in the GTFO SUDO tests. Had this command been instead “sudo /bin/bash” or even just “sudo -i”, the attack would have worked perfectly. It is difficult to draw any conclusive results from this, but it could indicate that with a gentle nudge by a human in the loop, this model could be quite effective. However, the biggest takeaway from these tests is that the a RAG-supported local LLM can perform successful privilege escalation attacks, which is a significant expansion on the previous study.

V. CONCLUSION

Powered by the rapid advancement of artificial intelligence, this research explores the feasibility of using generative AI technologies to carry out automatic cyber-attacks based on AI-generated system commands and their outputs. Our experiments demonstrate that large language models fine-tuned and augmented with security hacking information are promising. In particular, the process of adding RAG to a local LLM for penetration testing and integrating it in a localized environment provides a proof-of-concept that autonomous hacking is theoretically possible to successfully implement on average user machines. Future research is necessary to determine whether the models currently publicly available are well suited to this task and whether their performance can be improved through hardware, software, and data advancements.

REFERENCES

- [1] A. Happe and J. Cito, “Understanding hackers’ work: An empirical study of offensive security practitioners,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, New York, NY, November 30 2023, pp. 1669–1680.
- [2] D. Devadiga, G. Jin, B. Potdar, H. Koo, A. Han, A. Shringi, A. Singh, K. Chaudhari, and S. Kumar, “Gleam: Gan and llm for evasive adversarial malware,” in *14th International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, Korea, October 11-13 2023, pp. 53–58.
- [3] T. Naito, R. Watanabe, and T. Mitsunaga, “Llm-based attack scenarios generator with it asset management and vulnerability information,” in *6th International Conference on Signal Processing and Information Security (ICSPIS)*, Dubai, UAE, November 8-9 2023, pp. 99–103.
- [4] M. Sultana, A. Taylor, L. Li, and S. Majumdar, “Towards evaluation and understanding of large language models for cyber operation automation,” in *IEEE Conference on Communications and Network Security (CNS)*, Orlando, FL, October 2-5 2023, pp. 1–6.
- [5] M. Gupta, C. Akiri, K. Aryal, E. Parker, and L. Praharaaj, “From chatgpt to threatgpt: Impact of generative ai in cybersecurity and privacy,” *IEEE Access*, vol. 11, pp. 80 218–80 245, August 2023.
- [6] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, “Pentestgpt: An llm-empowered automatic penetration testing tool,” *arXiv preprint arXiv:2308.06782*, 2023.
- [7] K. Shashwat, F. Hahn, X. Ou, D. Goldgof, L. Hall, J. Ligatti, S. R. Rajgopalan, and A. Z. Tabari, “A preliminary study on using large language models in software pentesting,” *arXiv preprint arXiv:2401.17459*, 2024.
- [8] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier et al., “Mistral 7b,” *arXiv preprint arXiv:2310.06825*, 2023.
- [9] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-rank adaptation of large language models (poster),” in *International Conference on Learning Representations (ICLR)*, virtual, April 25-29 2022.
- [10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *Proceedings of the 34th Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, December 6 - 12 2020, p. 9459–9474.
- [11] A. Happe, A. Kaplan, and J. Cito, “Llms as hackers: Autonomous linux privilege escalation attacks,” *arXiv preprint arXiv:2310.11409*, 2024.
- [12] E. M. Kowira, N. N. Suki, and Y. Nathan, “Automated privilege escalation enumeration and execution script for linux,” *AIP Conference Proceedings on the International Virtual Conference on Machine Learning Applications in Applied Sciences and Mathematics (IVCMASM 2022)*, vol. 2802, no. 1, January 25 2024.
- [13] 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 (ESEC/FSE)*, New York, NY, November 30 2023, p. 2082–2086.
- [14] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, “Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning,” in *IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, Florence, Italy, October 9-12 2023, pp. 647–658.
- [15] O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgaay, A. Shashua, K. Leyton-Brown, and Y. Shoham, “In-context retrieval-augmented language models,” *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1316–1331, November 13 2023.
- [16] oobabooga et al., “A gradio web ui for large language models,” 2024. [Online]. Available: <https://github.com/oobabooga/text-generation-webui>
- [17] A. Happe, Neverbolt, and I. E. Ashimine, “Hackingbuddygpt,” 2024. [Online]. Available: <https://github.com/ipa-lab/hackingBuddyGPT>
- [18] R. Chandel, “Hacking articles,” <https://www.hackingarticles.in>, accessed: 2024-05-02.
- [19] “Hacktricks,” <https://book.hacktricks.xyz>, accessed: 2024-05-02.