# Hacking, The Lazy Way: LLM Augmented Pentesting

Dhruva Goyal, Sitaraman Subramanian, Aditya Peela

BugBase Pte Ltd

{dhruva, sitaraman, aditya} @bugbase.ai

Abstract—In our research, we introduce a new concept called "LLM Augmented Pentesting" demonstrated with a tool named "Pentest Copilot," that revolutionizes the field of ethical hacking by integrating Large Language Models (LLMs) into penetration testing workflows, leveraging the advanced GPT-4-turbo model. Our approach focuses on overcoming the traditional resistance to automation in penetration testing by employing LLMs to automate specific sub-tasks while ensuring a comprehensive understanding of the overall testing process.

Pentest Copilot showcases remarkable proficiency in tasks such as utilizing testing tools, interpreting outputs, and suggesting follow-up actions, efficiently bridging the gap between automated systems and human expertise. By integrating a "chain of thought" mechanism, Pentest Copilot optimizes token usage and enhances decision-making processes, leading to more accurate and context-aware Additionally, outputs. implementation of Retrieval-Augmented Generation (RAG) minimizes hallucinations and ensures the tool remains aligned with the latest cybersecurity techniques and knowledge. We also highlight a unique infrastructure system that supports in-browser penetration testing, providing a robust platform for cybersecurity professionals. Our findings demonstrate that LLM Augmented Pentesting can not only significantly enhance task completion rates in penetration testing but also effectively addresses challenges, marking a substantial advancement in the cybersecurity domain.

# I. Introduction

In the domain of cybersecurity, the automation of penetration testing (pentesting) represents a significant yet challenging milestone. Historically, the effectiveness of pentesting has been predominantly dependent on the skill and knowledge of human experts. This dependence has led to a bifurcation in the quality and cost of pentesting services: high-end, manual pentests are prohibitively expensive for many organizations, while more affordable options often rely on rudimentary automated tools that offer compliance rather than comprehensive security assurance. This dichotomy has contributed to a landscape where many organizations remain vulnerable to data breaches and regulatory penalties, despite ostensibly fulfilling pentesting requirements.

LLM Augmented Pentesting is a novel approach to this problem, aiming to democratize access to effective pentesting. [1, 6] By leveraging advanced LLMs, we can assist penetration testers in enhancing

their workflow efficiency. Pentest Copilot facilitates routine tasks such as documentation lookup, tool orchestration, and exploration of potential exploit strategies. Its unique contribution lies in its ability to automate aspects of the pentesting process without sacrificing the nuanced understanding and adaptability that human experts provide [3].

This research paper introduces details how the LLM Augmented Pentesting approach and how Pentest Copilot synergizes the vast, general-purpose knowledge bases of LLMs with the specific requirements of pentesting.

#### II. METHODOLOGY

# A. Large Language Model

## 1. Selecting a class of models

Our research indicated that among various LLMs available, the GPT lineage of models from OpenAI stood out for its extensive knowledge on security toolings and use cases [5, 6]. Its global knowledge, large training dataset, encompassing a wide array of cybersecurity papers and topics.

Our goal was to leverage a well-trained model capable of addressing cybersecurity and pentesting queries through guided instructions, before we proceeded to finetune our own model.

#### 2. Surface Evaluation of GPT Models

We evaluated the GPT lineage of models starting out with GPT-3.5-Turbo, GPT-4 and GPT-4-turbo [7, 8].

GPT-3.5-Turbo: While GPT-3.5-Turbo stands out for its speed, it falls short in accuracy, particularly in complex pentest scenarios. The model struggles with maintaining context over extended interactions, a critical factor in pentesting where referencing previous events or findings is essential due to less context size (4,096 tokens). Another limitation is lack of knowledge in general since the model is trained on the events up till September 2021, having access to latest practices and application of tooling is affected due to this. These limitations make it less suitable for nuanced and detailed pentest environments.

**GPT-4:** GPT-4 marks a significant improvement in response accuracy. It excels in handling complex pentest scenarios, including understanding and referencing previous events within a pentest sequence

1

due to increased context size (8,192 tokens) and also being trained on events up till April 2023. While it delivers in-depth and accurate information while handling previous context of the pentest engagement, the time taken for processing and generating responses is noticeably slower compared (refer to *Fig.* 2) to its turbo variant.

**GPT-4-turbo:** GPT-4-Turbo emerges as the currently best-suited model for pentest applications. It overperforms GPT-4 on context handling by being able to handle up-to 128k tokens which results in retrieving relevant commands and responses, the model is also 2.5x faster on average in returning responses (refer to *Fig. 3*).

# 3. Test Benching Models

We created a testbenching framework which mimics real-world pentesting scenarios. This framework included a range of activities from vulnerability assessments to threat modeling, across various network setups and systems. This involved hosting a 'boot2root box' [11, 12, 13], a purposefully vulnerable server, to test the capability of our models in achieving Remote Code Execution (gaining access to the server). Our approach spanned the entire spectrum of pentesting stages, from initial reconnaissance to the post-exploitation phase, providing a comprehensive and realistic assessment of the tool's effectiveness.

Evaluating the model response had to be done keeping in mind the following criterias:

- Accuracy: Correctness of the technical information, recommendations provided which included the commands to run, to the given testing scenario.
- **Response Structure:** Correctness of the response format which was a predefined JSON structure with exact parameters.
- **Response Time:** Speed of generating responses.

Below is the performance graph depicting how the models performed on the above criterias.

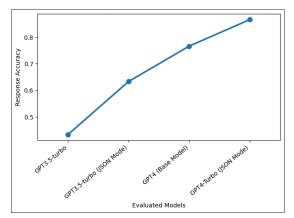


Fig 1. Analysis graph for evaluation of response accuracy when provided the same set of prompts to

each model. The x-axis represents all evaluated models, y-axis represents the average response accuracy for 10 test cases. (GPT-4-Turbo is the most accurate with a score of 0.87)

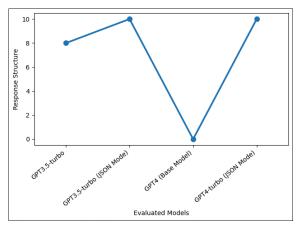


Fig 2. Analysis graph for evaluation of response accuracy when provided the same set of prompts to each model. The x-axis represents all evaluated models, y-axis represents the number of times a correct structure was returned for 10 test cases. (GPT-4-turbo and GPT-3.5-turbo with JSON mode enabled perform best 10/10 times)

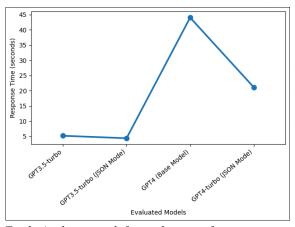


Fig 3. Analysis graph for evaluation of response time when provided the same set of prompts to each model. The x-axis represents all evaluated models, y-axis represents the average response time in seconds for 10 test cases. (GPT-3.5-Turbo with JSON Mode enabled is the fastest)

We can observe that GPT-4-turbo with JSON Mode enabled outperforms all other GPT lineage models in responding with accurate commands and generating responses with a correct structure. GPT-3.5-turbo is the fastest in response time but due to lesser accuracy in the responses, we decided to go ahead with GPT-4-turbo.

# **B.** Prompt Engineering

Prompt engineering plays an important role in LLM Augmented Pentesting, ensuring that the model understands and responds to cybersecurity-related queries with precision. By crafting specific,

context-aware prompts, the model can interpret technical language and ethical hacking related terminologies effectively [5, 6].

## 1. Jailbreaking GPT

By design, GPT is not configured to specifically cater to pentesting queries. Its general-purpose design meant that it provided broad information across various domains, but due to security reasons and constraints placed by OpenAI [7, 8], the model would not respond to a pentesting query.

The key was to instruct GPT to assume the role of a "Penetration testing assistant, collaborating with a security researcher". [4] The instruction was given to GPT in its system prompt which is a constraint that it will follow at all times while returning a response. The different types of prompts are elaborated in the "Dissecting Prompts" section.

This shifted the model's operational framework from a **generalist** to a **specialist**, focusing its vast knowledge base specifically towards pentesting.

# 2. Dissecting Prompts

We have utilized the Chat Completions API of the GPT4-turbo model. The primary input for Chat Completions API is the "messages" parameter, which is an array of message objects. Each message object has a "role" (either "system," "user," or "assistant").

# • System/Instruction Prompt

The system prompt at the initiation of Pentest Copilot explicitly outlines the objectives, enumerates constraints, and provides context regarding the configured tools for utilization in commands. This was crucial to prevent inadvertent tool name hallucinations and inaccurate information by the LLM.

# • User Prompt

In the user prompt, we dynamically adjust the instructions based on whether the user provides previous testing information for the pentest target or not. If the user provides previous testing details, we skip adding the prompt for the preliminary testing phase. However, if no previous testing information about the target is provided, the prompt with either a Domain or IP scan is added. The prompt also outlines the expected JSON response format for output and sets constraints, such as recommending a single command if possible. Emphasis is placed on prioritizing user input to ensure the pentest aligns with their preferences, avoiding autonomous decisions by the model. Additionally, if no target information is provided, Pentest Copilot asks for more details about the target.

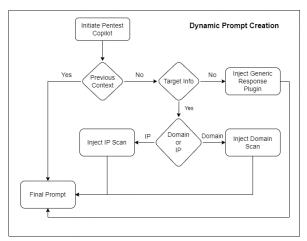


Fig. 4 - Flowchart on Dynamic Prompt Creation where new target information and previous context is injected into the user prompt based on the current state of pentest

For tasks such as executing commands and creating payloads, we've specified a set of plugins directly in the prompt. Here's an overview of some of these plugins:

- 1. **Web Search Plugin:** Enables the LLM to conduct web searches for discovering new exploits or methodologies.
- 2. **Run Bash Plugin:** Allows the LLM to execute bash commands including open source pentest tools for various tasks.
- 3. **Generic Response Plugin:** This plugin gives a text response that contains insights from the LLM when specific tasks cannot be completed using plugins. Moreover, Pentest Copilot can execute this plugin to request additional information from the user.
- 4. **Netcat Listener Plugin:** Initiates a Netcat listener on a specified port, creating a listener for a reverse shell connection.
- 5. **Generate Payload Plugin:** Generates an exploit payload using the metasploit framework [9].

These plugin specifications are incorporated directly into the prompt, providing clear instructions to the model on when and how to employ each plugin.

#### 3. Prompt Bifurcation and Context length

In developing Pentest Copilot as an ethical hacking assistant, one of the significant challenges we faced was managing the context length limitation of 4,096 tokens inherent in the GPT-3.5 model. This limitation posed a unique challenge, especially considering the complexity and depth required for effective pentesting assistance. Our approach to this challenge involved innovative strategies to optimize token usage and chain various steps efficiently.

Our initial setup included various essential components, each consuming a significant portion of the token limit:

- 1. **Instruct System Prompt (250 tokens):** The foundational prompt to guide the AI to act as an ethical hacking assistant.
- 2. **Plugin Information Prompt (600 tokens)**: Descriptions of all available plugins and their functionalities.
- 3. Command Response Format Specification (500 tokens): Guidelines on the format in which responses should be structured.
- 4. **JSON Response Format Constraints (500 tokens)**: Constraints set for responses to ensure consistency and parsability.
- 5. Pentest Task and Previous Contexts (600 tokens): Details of the current pentesting task along with relevant historical data.
- 6. **To-Do Checklist (800 tokens)**: A persistent list of tasks and objectives to be maintained throughout the pentest.
- 7. **Updating the To-Do Checklist (800 tokens)**: Dynamic modifications to the to-do list based on ongoing activities.
- 8. Summarizing current pentest state (500 tokens): Summarization of each executed command for clarity and record-keeping.

These components, essential for the functioning of Pentest Copilot, collectively exceeded the token limit, necessitating a more efficient approach.

We tackled the problem of exceeding the token limit per API call by **Step Chaining** - explained in detail below.

# C. Step Chaining Approach for Token Size Optimization

To address the constraint of maximum tokens per API call, we adopted a strategy of step chaining similar to the chain of thought approach [2], which involved breaking down the process into distinct phases:

- 1. **Command Generation**: Generating specific commands or actions for the pentest, tailored to the current context and objectives.
- 2. Summarization of Current State: Concisely summarizing the state of the pentest after each command, including outcomes and any changes in the scenario.
- 3. **Updation of a To-Do List**: Reflecting these changes in the to-do list, ensuring it remains current and relevant.

By chaining these steps together, we could maintain a continuous flow of relevant information without exceeding the token limit. This method allowed us to allocate tokens more judiciously, ensuring that each component received enough space to be effective without being overly verbose.

The step chaining process (refer Fig. 5) also ensured that the model kept a coherent and relevant context throughout the pentesting session, crucial for maintaining the quality and applicability of its guidance.

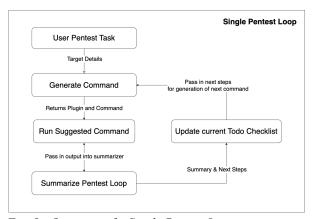


Fig. 5 - Overview of a Single Pentest Loop covering the task from the user, generation of commands, running suggested commands followed by summarizing the pentest loop and updating the todo-list

# D. Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG) plays a pivotal role in enhancing LLM Augmented Pentesting, particularly in addressing the challenge of providing up-to-date and accurate information on cybersecurity tools and practices [15].

One significant challenge in developing the tool was ensuring that the LLM had access to the latest information on the usage, syntax, and modules of complex cybersecurity tools and frameworks like Metasploit [2]. The dynamic nature of these tools, with frequent updates and new modules, means that any static knowledge base quickly becomes outdated.

To prepare the vector database for RAG in Pentest Copilot, we collected and curated a comprehensive dataset of the latest modules, scripts, and usage guides specifically from Metasploit and MSFVenom. The dataset included structured information such as command syntax, use cases, and detailed documentation for these tools. This information was then processed to convert the textual data into vector representations, or embeddings, that capture the semantic meaning of the content. These embeddings were indexed and stored in the vector database, allowing for efficient similarity searches and quick retrieval of the most relevant and up-to-date information during LLM-driven pentesting sessions.

How RAG works in the system:

1. **Query Processing**: When a user queries about a specific module or syntax, the LLM processes this query to understand the context and the specific information

required.

- 2. **Information Retrieval**: The LLM then uses RAG to reach out to an external knowledge base, seeking the most relevant and current information related to the query.
- 3. **Data Integration**: The retrieved information is integrated with the LLM's internal knowledge to generate a comprehensive, formatted, accurate response.

# E. Preferred Tooling and Context

In the LLM Augmented Pentesting approach, a researcher may prefer to use tools and techniques they are familiar with. To accommodate this, users can select their preferred tools, and these choices are then integrated into the system prompt.

This customization allows the model to prioritize the user's chosen tools when suggesting commands. Additionally, the system prompt is designed to recommend alternative tools when deemed necessary. In such cases, the response provides a clear explanation for these recommendations, ensuring the user understands why an alternative tool might be more effective in a specific situation.

# F. File Analysis with LLMs

In pentesting, researchers frequently encounter files that, when analyzed, can lead to new discoveries. However, LLMs cannot directly understand the content of these files. Therefore, the information within these files must first be converted into a plaintext format that LLMs can interpret.

The file analysis tool first checks the file format using the Linux "file" command. It then runs the appropriate analysis procedure based on the output of the file command. The tool also scans media files for any hidden file signatures, and recursively extracts and analyzes any additional files found.

The tool is currently designed to deal with such files by including large lines of text from the file in the output, limited by the allowed number of tokens to OpenAI's API  $[\underline{7}, \underline{8}]$ .

For different file formats, the following contextual data is extracted:

# 1. ELF:

- a. Security Settings of the file:
  - i. NX bit
  - ii. RELRO
  - iii. Stack Canary
  - iv. PIE
- b. stdlib functions used in the file.
- c. Symbols present in the binary.
- d. Link Tye: Dynamic/Static.
- e. Whether the binary is a shared object or not.
- f. File Architecture.

#### 2. PE32/PE32+:

- a. Dependencies of the executables.
- b. Number of sections in the executable.
- c. Security Settings present in the executable:
  - i. Non-Executable Bit.
  - ii. Stack Canary.
  - iii. Structured Exception Handling.
- d. Name of Libraries used by the executable.
- e. Imported/Exported functions.

#### 3. XML/JSON/YAML:

For the configuration files, a minimalistic tree of all the parents and their possible child elements is created and structured to represent the hierarchical relationship between them.

#### 4. Media Files:

- a. Exif data.
- b. Any hidden signatures of other file types.

# 5. Plain Text Files:

- a. A python3 library "Guesslang" is used to detect any programming language. Depending on the detected language, function names are then extracted using regex [16].
- b. If no language is accurately detected, only very large lines (> 100 words) are extracted and sent with the context.

The list of files that the File Analysis tool can analyze is currently limited to the following list:

- **Binary files** ELF and PE32+ file formats for Linux and Windows
- Media Files MP3,MP4, PDF, PNG, JPEG etc.
- Configuration Files XML, OpenVPN, JSON and YAML.

# III. INFRASTRUCTURE

# A. Setting up Infrastructure

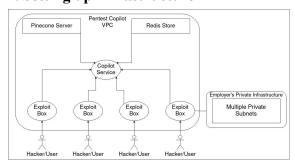


Fig. 6 - Complete infrastructure of Pentest Copilot -All services and Exploit Boxes are in a single VPC (Virtual Private Cloud), each container can communicate with each other

Pentest Copilot necessitates a platform that allows researchers to conduct seamless pentesting sessions with its assistance. Interruptions or connectivity issues can hinder the effectiveness of the testing process which is dodged by dynamically setting up cloud resources, specifically, spinning up an on-demand server instance called "Exploit Box" for each session.

- 1. This ensures dedicated and isolated environments for every researcher. The Exploit Box houses a plethora of tools to aid the researcher. (Refer to Table 1)
- 2. The use of graphical user interface (GUI) tools such as Burp Suite is crucial and common in pentesting. Enabling a desktop experience becomes essential for using these tools to be installed by researchers. To address this, we incorporate a full desktop experience through VNC (Virtual Network Computing), allowing users to seamlessly utilize GUI tools like Burp Suite within the pentesting environment.
- 3. Managing session-specific data, including user files and artifacts, across multiple testing sessions on the same target is essential to ensure continuity of pentesting sessions. Ensuring persistence and accessibility of these files is crucial for an efficient workflow. To address this, we can implement a structured data storage solution. Each session is assigned a dedicated volume, providing a persistent storage unit. This volume is attached to the Exploit Box launched for that specific session, allowing users to store and retrieve session-specific data seamlessly.

# B. Choosing the right security tools

The decision to integrate multiple open-source tools into Pentest Copilot was based on their open source nature and widespread use. For example, Ffuf is being used for directory brute forcing which provides the flexibility to choose the payload position and auto calibration for determining the baseline error response. These tools are frequently utilized by bug bounty hunters and security researchers to perform tasks essential to their work.

TABLE I : LIST OF OPEN SOURCE TOOLS USED IN PENTEST COPILOT

Tool Name	Forks	Stars	Use Case
Ffuf	1.2k	10.8k	
Feroxbuster*	422	5k	
Dirsearch	2.3k	10.8k	Directory Brute Forcing
Wfuzz	1.3k	5.5k	-
Gobuster	1.2k	8.6k	
Subfinder*	1.2k	8.8k	

Tool Name	Forks	Stars	Use Case	
Amass	1.8k	10.7k	Subdomain Discovery Passive	
Findomain	367	3.1k		
Assetfinder	467	2.7k		
Shuffledns	178	1.2k	Subdomain Brute Forcing	
Puredns	146	1.5k		
dnsx	235	1.8k		
THC-Hydra*	1.9k	8.7k		
Patator	788	3.4k	Credentials Brute Forcing	
Crowbar	328	1.3k		
Wpscan	1.3k	8.1k	CMS Scanners	
Drupwn	134	549		
CMSmap	249	939		
Nmap*	2.3k	8.8k	Port Scanning	
Naabu	482	4k		
Smap	234	2.7k		
Masscan	3.3k	22.2k		
WayBackUR Ls	434	3k	Spider/Crawling	
Gau	401	3.3k		
xnlLinkFinde r	130	963		
Waymore	146	1.2k		
Katana	434	3k		
Gospider	337	2.3k		
Whatsweb	906	5k	Fingerprinting Technologies	
Httpx	756	6.4k		
Sqlmap*	5.6k	29.6k	Vulnerability Exploitation	
Ghauri	201	1.7k		
GraphQLma p	188	1.2k		
Dalfox	379	3.1k		
SecretFinder	331	1.6k		

<sup>\*</sup>Default tools used in Pentest Copilot

#### C. Socket communication with instances

Pentest Copilot utilizes sockets for real-time communication between the user, copilot

infrastructure, and the exploit box server (refer *Fig.* 7).

Here's an overview of how it functions:

- 1. **Initiation of communication:** When a user starts an exploit box, a socket communication channel is established between the user (client side) and the infrastructure. In parallel, a Secure Shell (SSH) connection is formed between the infrastructure and the exploit box server. This setup ensures a secure and seamless flow of information.
- 2. Command Processing and Execution: An interactive terminal-like interface is displayed to the users. Here, the user can input commands into the terminal, the input data is transmitted to the infrastructure via the socket connection. Commands received from the client are forwarded by Pentest Copilot to the exploit box server using SSH. Pentest Copilot maintains continuous communication with the exploit box to capture any feedback or results from the executed commands.
- Displaying Results: The results and outputs from the exploit box are sent back to the client side via the Pentest Copilot infrastructure, ensuring that the user receives immediate feedback and results displayed on their terminal interface.

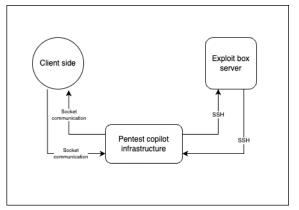


Fig. 7 - Two-way socket communication between client and exploit box server with Pentest Copilot Server in the middle for relaying communication

# **D. VPN Configuration**

Pentest Copilot is also capable of targeting entities located on private subnets. There are many controls put into place to ensure all parties interacting with Pentest Copilot can comfortably partake in a pentesting engagement be it professional, or for research purposes.

 Pentest Copilot has been engineered to connect to private subnets through an established OpenVPN server. This ensures

- secure and controlled access to the private network.
- 2. Users may establish a secure connection to their home subnets using a tool called *Chisel* in case they don't have a VPN server setup for themselves. This ensures flexible and secure communication between Pentest Copilot and the researcher's home lab, supporting both professional pentesting and research in diverse settings.

# E. Mitigating possible malicious use

A careful consideration is taken while creating security policies preventing researchers from potentially abusing the cloud resources allocated to them during testing. This is essential for maintaining the integrity of the testing environment and ensuring fair and ethical usage.

TABLE II: RISK AND MITIGATION ANALYSIS

Туре	Risk	Mitigation			
Malicious use of cloud resources	Password cracking or Crypto Mining on Exploit Box infrastructure	CPU Usage Monitoring and Execution caps on resources.			
Malicious use of Pentest Copilot	Users targeting systems which one is not authorized to test on from Exploit Box instance	No direct traffic egress to public internet			
Exploitation of Pentest Copilot	Leaked API keys	Secrets Management and Key Rotation			
Exploitation of Pentest Copilot	RCE on Pentest Copilot instance	Rigorous and Regular Pentesting and Remediation			
In-Between Users Exploitation	Two users on the same internal subnet, while using Exploit Box, can target each others systems and data	Restricted VPS			

# IV. RESULTS

# A. Performance

The journey of Pentest Copilot's development has seen remarkable improvements in performance, particularly in the areas of response time, accuracy, and functionality. These enhancements have significantly elevated the tool's utility for pentesters, streamlining their workflow and providing more precise assistance.

Originally, Pentest Copilot took an average of **8-10 minutes** to perform a series of critical tasks:

- 1. **Generate a Command**: Crafting a specific command relevant to the current stage of the pentest.
- 2. Summarize the Current Pentest State: Providing a concise overview of the pentest's current status.
- 3. **Update the To-Do List**: Reflecting new tasks or findings in the pentest checklist.

This duration was a bottleneck, especially in scenarios where quick decision-making and rapid response were crucial.

The introduction of GPT-4-Turbo and the optimization of our prompting techniques marked a turning point. With GPT-4-Turbo's enhanced processing speed and our efficient prompt design, the time required for the above tasks has been nearly **halved to about 4-5 minutes**. Furthermore, the incorporation of JSON Mode within GPT-4-Turbo streamlined the data handling process, further contributing to this time reduction.

The integration of RAG has brought a new level of accuracy to the commands generated by Pentest Copilot. By accessing the most current and relevant information from a vast external knowledge base, RAG ensures that the commands and advice provided are not only timely but also highly precise and tailored to the latest cybersecurity practices.

A significant addition to Pentest Copilot is the introduction of file analysis functionality. This feature enables pentesters to efficiently analyze findings and data collected during a pentest. It simplifies the process of sifting through complex datasets, logs, or reports, allowing for quicker identification of vulnerabilities, anomalies, or patterns. This capability is particularly valuable in post-pentest reviews, where thorough analysis is key to understanding the effectiveness of the pentest and planning future security strategies.

### **B.** Shortcomings

Pentest Copilot's current capabilities are limited to the underlying LLM. This constraint implies that the system may not fully comprehend or execute complex operations associated with sophisticated testing tools or methodologies, potentially hindering its performance in scenarios demanding advanced capabilities beyond the scope of the LLM. These limitations include, but are not limited to:

- 1. Not being able to mount zero-day attacks
- 2. Unable to bypass security controls which are not known to the LLM
- 3. Using attack patterns which may have been overused a lot resulting in easy detection by security systems.

At present, our approach lacks in-depth awareness of the intricate operations of closed-source and paid frameworks/tools. For example, the tool is currently unable to properly instruct a new Cobalt Strike user on how to set up Beacons, Proxies and mount lateral traversal attacks and correlate each Beacon's position and importance in the network. The platform's capabilities may be constrained in dealing with the specific functionalities and nuances of these tools, potentially limiting its effectiveness in certain advanced testing scenarios which touch upon red teaming for example, malware writing, bypassing EDR (Endpoint Detection and Response) or advanced Active Directory attack scenarios.

# V. Conclusion

Pentest Copilot is set to advance by integrating with diverse knowledge bases for technology-specific expert advice. Future versions will feature fine-tuned models for pentesting and red teaming activities, enhancing capabilities in exploit writing and understanding closed-source industry tools in the pentesting and red-teaming spaces. The LLM Augmented Pentesting approach will involve maintaining a dynamic buffer of identified misconfigurations during targeting, allowing real-time adaptation and pattern recognition.

Additionally, a red team-specific knowledge base will be developed, covering both non-technical security issues (e.g., phishing) and technology-heavy misconfigurations (e.g., Group Policy Objects misconfigurations in Active Directory). These strategic enhancements aim to make Pentest Copilot an agile, expert, and indispensable tool for ethical hackers and penetration testers.

### ACKNOWLEDGMENTS

The authors would like to thank Siddharth Johri for his work on VPN configuration and deployment, and Devang Solanki and Tuhin Bose for their contributions to security and functionality testing, including identifying errors in the solution. We also extend our gratitude to Bhavarth Karmarkar for his efforts in developing the file analysis solution.

The authors would like to express their gratitude to the developers and maintainers of the various open-source tools that were made available to Pentest Copilot. Although we did not use these tools directly in our research, they played a crucial role in enabling the framework to access a wide range of capabilities for comprehensive security testing and analysis

- Directory Brute Forcing: <u>ffuf</u>, <u>Feroxbuster</u>, <u>Dirsearch</u>, <u>Wfuzz</u>, and <u>Gobuster</u>
- Subdomain Discovery (Passive): <u>Subfinder</u>, <u>Amass</u>, <u>Findomain</u>, and <u>Assetfinder</u>
- Subdomain Brute Forcing: <u>Shuffledns</u>, <u>Puredns</u> and <u>dnsx</u>
- Credentials Brute Forcing: <u>THC-Hydra</u>, Patator and <u>Crowbar</u>

- CMS Scanners: <u>Wpscan</u>, <u>Drupwn</u> and <u>CMSmap</u>
- Port Scanning: <u>Naabu</u>, <u>Smap</u>, <u>Masscan</u>, and Nmap,
- Spidering/Crawling: <u>Gau</u>, <u>xnLinkFinder</u>, <u>Waymore</u>, <u>Katana</u>, <u>Gospider</u>, and <u>waybackurls</u>
- Fingerprinting Technologies: <u>Httpx</u> and WhatWeb
- Vulnerability Exploitation: <u>Sqlmap</u>, <u>Ghauri</u>, <u>GraphQLMap</u>, <u>Dalfox</u>, <u>SecretFinder</u>

# REFERENCES

- Tann, W., Liu, Y., Sim, J. H., Seah, C. M., & Chang, E. C. (2023). Using large language models for cybersecurity capture-the-flag challenges and certification questions. arXiv preprint arXiv:2308.10443.
- [2] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35, 24824-24837.
- [3] Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.(2023). arXiv preprint arXiv:2302.06590.
- [4] Liu, Y., Deng, G., Xu, Z., Li, Y., Zheng, Y., Zhang, Y., ... & Liu, Y. (2023). Jailbreaking chatgpt via prompt engineering: An empirical study. arXiv preprint arXiv:2305.13860.

- [5] Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., ... & Rass, S. (2023). Pentestgpt: An Ilm-empowered automatic penetration testing tool. arXiv preprint arXiv:2308.06782.
- [6] Happe, A., & Cito, J. (2023, November). 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 (pp. 2082-2086)
- [7] "Models openai api," https://platform.openai.com/ docs/models/, (Accessed on 02/02/2023).
- [8] "Gpt-4," https://openai.com/research/gpt-4, (Accessed on 06/30/2023).
- [9] Rapid7, "Metasploit framework," 2023, accessed: 30-07-2023. [Online]. Available: <a href="https://www.metasploit.com/">https://www.metasploit.com/</a>
- [10] "OWASP Juice-Shop Project," https://owasp.org/ www-project-juice-shop/, 2022.
- [11] "Hackthebox: Hacking training for the best." [Online]. Available: https://www.hackthebox.com/
- [12] "TryHackMe: TryHackMe is a free online platform for learning cyber security, using hands-on exercises and labs, all through your browser!" [Online]. Available: <a href="https://tryhackme.com/">https://tryhackme.com/</a>
- [13] [Online]. Available: https://www.vulnhub.com/
- [14] "OWASP Foundation," https://owasp.org/
- [15] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.
- [16] Yoeo. (2024). Guesslang: Detect the programming language of a source code. GitHub. https://github.com/yoeo/guesslang