2025 IEEE International Conference on Future Machine Learning and Data Science (FMLDS) | 979-8-3315-5397-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/FMLDS67896.2025.00135

LIMA: Leveraging Large Language Models and MCP Servers for Initial Machine Access

Mohammad Ibrahim Saleem[1], Sohan Simha Prabhakar[1], Abhinav Harsha[2], Devayani Nagabhushan[3], William Arthur Conklin[1], Kyu In Lee[1], and Tania Banerjee[1]
[1]Department of Information Science Technology, University of Houston, Houston, Texas 77204
[2]Top Grep Tech, Bangalore, India
[3]Lebanon Trail High School, Frisco, Texas 75035
Email: {msaleem10, sprabhakar3, waconklin, klee49, tbanerjee}@uh.edu, {abinavharsha, devayani.nagabhushan}@gmail.com

***Abstract*—We present LIMA (Large Language-Model-driven Initial Machine Access), a penetration testing framework that pairs off-the-shelf LLMs with Model Context Protocol (MCP) servers to orchestrate automated initial-access reconnaissance, enumeration, and exploitation. LIMA's modular design allows an LLM to invoke scanners, parse outputs, consult vulnerability databases, and generate exploits while receiving minimal human input. We evaluate Claude 3.5 and GPT-4o on five targets: four HackTheBox machines and a custom VM and compare their performance with that of beginner- and expert-level human testers. Claude finished four boxes in a mean time of 13 minutes which is up to 2x faster than the expert and required help only for CAPTCHAs, achieving autonomous completion otherwise. GPT-4o, with sporadic guidance, completed two boxes in 17.5 minutes. Both models failed on a BurpSuite-dependent target, showing that missing GUI tooling, not model logic, is now the main bottleneck. Token-cost analysis puts a full autonomous run at $\leq$ $0.05. These results establish the first quantitative baseline for AI-augmented penetration testing at the initial-access phase.**

***Index Terms*—Large Language Models, Initial Access, Cybersecurity Reconnaissance, Prompt Engineering, AI-driven Cybersecurity.**

I. Introduction

In penetration testing, initial access to a target system is the most critical phase of any cyber-attack [1], forcing an adversary to bridge the gap between external reconnaissance and internal compromise. This step of the cyber kill chain typically demands extensive technical expertise: attackers must identify vulnerable services, understand their weaknesses, craft appropriate exploits, and execute them while evading detection [2], [3]. Yet the rapid advancement of large language models (LLMs), for example, PenTestGPT [4] and work by Happe and Cito [5] already interpret advisories, generate exploit code, and debug failures. This capability shift raises an urgent question echoed by ADAPT [6]: can artificial intelligence automate the expertise required for initial access?

Despite growing alarm over LLM-assisted attacks, no prior work has systematically measured how far an off-the-shelf model can *autonomously* drive an intrusion, namely, from reconnaissance all the way to initial access [5], [6]. Existing studies mostly discuss hypothetical risks or showcase one-off demos that still depend on heavy human steering [4], [7]. We close this gap with a controlled framework in which the LLM itself launches scans, selects exploits, and adapts to system feedback, while a human operator merely observes and intervenes only when strictly necessary e.g., when the process stalls on CAPTCHAs or similar roadblocks. By cleanly separating the model's reasoning from occasional safety interventions, we quantify its tactical capability without conflating it with manual shell work. Claude 3.5 and GPT-4o were evaluated against four HackTheBox machines covering remote-code execution, SQL injection, command injection, and credential disclosure and a custom exam VM, with results compared to beginner- and expert-level human testers under identical conditions.

Evaluating LLM capabilities in offensive security presents several challenges. First, the stochastic nature of LLM outputs creates reproducibility concerns, as identical prompts can yield different exploitation strategies [8]. Second, isolating the LLM's strategic reasoning from mechanical execution requires careful experimental design to avoid conflating planning with tooling effects [6]. Third, fair comparison with human testers must account for fundamentally different problem-solving modes [4]. To address these challenges, we develop a standardized evaluation protocol that ensures consistent testing conditions across all trials. Our framework implements structured prompt engineering to minimize output variability, employs strict operator guidelines to maintain execution consistency, and establishes clear metrics that capture both success rates and reasoning quality. This systematic approach allows us to quantify LLM effectiveness while acknowledging the inherent complexities of human-AI comparison in dynamic security scenarios.

Our empirical evaluation demonstrates clear differences in how LLMs approach initial access challenges compared to human penetration testers. The LLMs demonstrated strong performance in tasks requiring pattern matching and knowledge retrieval, successfully identifying vulnerable services and mapping them to known exploits in most cases. However, they exhibited critical weaknesses in adaptive problem-solving, particularly when initial exploitation attempts failed or when targets presented non-standard configurations. These findings suggest that while LLMs can accelerate certain aspects of penetration testing, they lack the creative reasoning and contextual understanding that experienced human testers employ. Our work provides the first quantitative baseline for understanding these capabilities and limitations, offering essential data for security teams preparing defenses against AI-augmented threats.

979-8-3315-5397-5/25/$31.00 ©2025 IEEE
DOI 10.1109/FMLDS67896.2025.00135

This work makes the following key contributions:

1) We introduce *LIMA*, the first modular system that pairs off-the-shelf large language models with Model Context Protocol (MCP) servers to automate reconnaissance, enumeration, and exploitation for initial access.
2) We create the first quantitative benchmark for AI-driven initial access, evaluating Claude 3.5, GPT-4o, and two human testers on four public HackTheBox machines plus a private exam VM. LIMA's modular design also allows other LLMs to be plugged in effortlessly.
3) Experiments show Claude 3.5 achieves 90–100 % autonomous completion on low-complexity boxes, while failures stem from missing GUI-bound tooling (e.g., BurpSuite) rather than LLM logic. We provide detailed token-usage and API-cost metrics.

The remainder of this paper is organized as follows: Section II reviews prior work in the domain, and Section III describes our methodology. Section IV presents the experimental findings, with concluding remarks provided in Section V.

II. Related Work

Early work showed that LLMs can act as "AI sparring partners," helping testers brainstorm attack plans and probe single, sandboxed targets under tight human guidance [5]. Follow-up studies went modular: a three-stage controller boosted task-completion rates by orchestrating tool calls and prompt chains [4], while a 50 k-entry question–answer corpus improved fine-tuned models on quiz-style penetration tasks [7]. Although encouraging, these efforts remain confined to narrow sub-tasks, scripted labs, or Q&A settings that stop short of the messy, end-to-end challenge posed by preactical initial access.

Beyond LLM-centric research, a rule-based architecture has automated attack planning along predefined paths on Metasploitable snapshots and a training lab [6]. In parallel, an interview study mapped how seasoned white-hat professionals make decisions and highlighted the need for human oversight in high-risk environments [9]. Both strands underscore important tooling and workflow insights but still assume either static exploit graphs or continuous human control.

The literature thus offers several pieces of the puzzle, including prompt-driven helpers, modular controllers, curated datasets, and formal attack graphs. What remains untested is the central question posed in our introduction: can an off-the-shelf LLM, without task-specific fine-tuning or custom modules, autonomously bridge reconnaissance and compromise across diverse live systems? Our system is designed to operate with minimal human intervention and, in many cases, can complete the entire initial access workflow 100% autonomously. By placing such a model in direct competition with human testers on four HackTheBox machines and isolating its reasoning from execution mechanics, our study supplies the first quantitative baseline for AI-driven initial access under conditions that resemble real enterprise targets.

III. Methodology

In this work, we adopt a structured penetration testing methodology to simulate real-world cyberattacks and evaluate the effectiveness of both human-led and LLM-assisted approaches for gaining initial access to vulnerable systems. Our five-phase penetration-testing framework is described in Section III-A. The LLM-assisted access pipeline and its MCP integration are outlined in Section III-B, while Section III-C details the Model Context Protocol (MCP) architecture that underpins LIMA. Finally, Section III-D presents the evaluation strategy, including the protocol, metrics, and test environment.

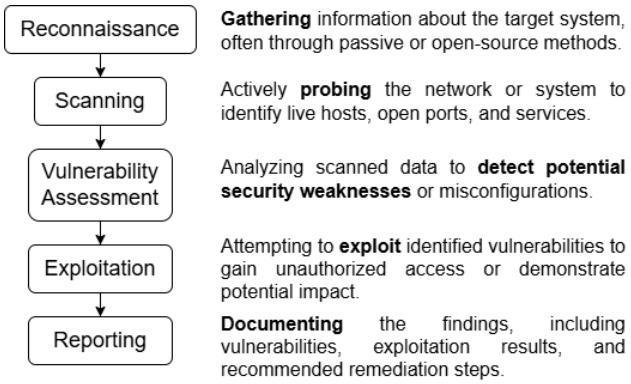

Fig. 1. The five-phase penetration testing methodology defined by the EC-Council.

A. Penetration Testing Framework

A leading cybersecurity certification body, the International Council of E-Commerce Consultants (EC-Council), has defined a structured penetration testing methodology [1] consisting of five distinct phases (reconnaissance, scanning, vulnerability assessment, exploitation, and reporting) as illustrated in Figure 1:

a) ***Reconnaissance****:* : The first step in the penetration testing process is gathering information about the target system or organization. This involves collecting details such as user accounts, operating systems, applications, and network topology. Reconnaissance can be conducted through two primary approaches: (a) passive methods, which rely on publicly available or open-source information without direct interaction with the target, and (b) active methods, which involve direct engagement with the target system to extract relevant data.

b) ***Scanning****:* : Scanning builds on the reconnaissance stage by actively probing the network or system to identify live hosts, open ports, and running services. This step uses various tools to track network activity on the target host. Identifying open ports is critical, as they often represent potential entry points for attackers and inform the development of effective penetration strategies.

c) ***Vulnerability Assessment****:* : In this stage, the tester analyzes the scanned data to detect potential security weaknesses or misconfigurations, using insights gathered during reconnaissance and scanning. To assess the severity and exploitability of identified vulnerabilities, resources such as the Common Vulnerabilities and Exposures (CVE) database and the National Vulnerability Database (NVD) are commonly consulted.

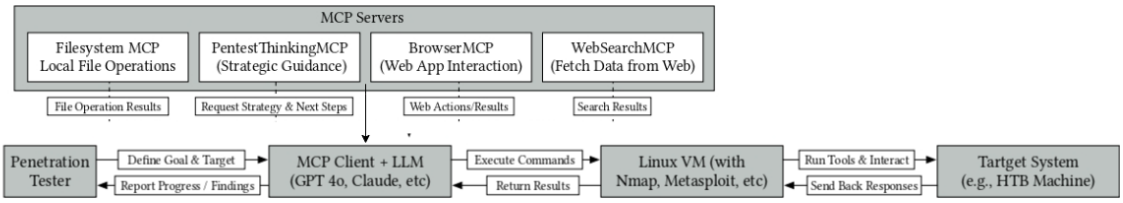

Fig. 2. Architecture of our penetration testing system integrating the Model Context Protocol (MCP), where an LLM (e.g., Cursor, Claude) operates in tandem with a pentester, a Linux VM with standard penetration tools (e.g., Nmap, Metasploit), and a set of MCP modules (FilesystemMCP, PentestThinkingMCP, BrowserMCP, WebSearchMCP). The MCP framework enables context-aware action planning, dynamic guidance, and multi-modal data integration to simulate real-world penetration testing workflows.

d) ***Exploitation****:* : Once vulnerabilities have been identified, the tester attempts to exploit them to gain unauthorized access or demonstrate their potential impact. This phase simulates real-world attacks to assess the effectiveness of the target system's security defenses. Tools such as Metasploit are often used to automate and execute these exploits.

e) ***Reporting****:* : The final phase involves documenting the findings of the penetration test in a comprehensive report. This includes (a) a summary of identified vulnerabilities along with their severity levels, (b) an assessment of the potential business impact, (c) detailed accounts of exploitation attempts and their outcomes, and (d) actionable recommendations for remediation and improving the organization's overall security posture.

This methodology simulates real-world cyberattacks to allow a systematic assessment of an organization's security posture. In this work, we apply it both for manually obtaining initial access and within the LLM-assisted pipeline.

B. LLM-assisted Access Pipeline

In parallel with the manual access baseline, we develop and evaluate an LLM-assisted software (LIMA) aimed at replicating the reasoning process of a human penetration tester. The pipeline leverages a general-purpose LLM to interpret reconnaissance outputs, suggest tools and techniques, and adapt based on real-time feedback during initial access attempts.

The LLM operates within a modular framework tailored to support the initial access phase of penetration testing. While the broader architecture leverages protocol-driven context management and tool integration (described in Section III-C), our focus here is on assessing the LLM's ability to autonomously perform key initial access tasks:

1) Interpret service banners and `nmap` output
2) Recommend enumeration strategies and appropriate tools
3) Identify or generate suitable exploit scripts based on discovered vulnerabilities
4) Adapt its recommendations based on system feedback from executed commands

Leveraging MCP servers and LLMs, our system autonomously performs approximately 90% of the initial access workflow, requiring only occasional human intervention for executing complex or privileged actions. This design enables realistic and controlled testing while preserving the model's capacity for iterative, tool-informed reasoning throughout the initial access phase.

C. Model Context Protocol (MCP)-Assisted Architecture

Our system architecture integrate human testers, a general-purpose large language model (LLM), and modular context servers for penetration testing tasks. MCP helps establish a standardized interface designed to let reasoning agents (such as large language models or autonomous AIs) interact with modular backends ("MCP servers") to perform tasks such as search, planning, tool execution. An MCP server serves as a pluggable tool that understands prompts, generates structured outputs, and can collaborate with AI agents in a reasoning loop.

Figure 2 illustrates the interaction between the pentester and the MCP client, which incorporates a large language model (e.g., GPT-4o, Claude) via an IDE with LLM-based support (e.g., Cursor) and oversees context across many data sources. The MCP client, integrated into the IDE, enables smooth interaction between the LLM and context-sensitive modules during penetration testing. It facilitates communication with a Linux virtual machine operating standard penetration testing tools (e.g., Nmap, Metasploit) and establishes connections to target systems (e.g., HackTheBox machines) for real-time testing. A collection of MCP servers—FilesystemMCP [10] for file management, PentestThinkingMCP [11] for strategic insights, BrowserMCP [12] for web engagement, and WebSearchMCP [13] for online data acquisition—delivers contextual information and facilitates dynamic, context-oriented decision-making. This hybrid architecture allows the machine to replicate human-like adaptation and reasoning while ensuring consistency across various inputs and activities during the penetration testing process.

While the generic MCP framework handles context exchange, effective offensive reasoning requires a domain-aware backend specifically tuned for penetration testing; the following subsections describes these steps.

1) Our Server: We developed a customized MCP server, the PentestThinkingMCP, that is designed to provide structured, context-aware guidance for penetration testing. Unlike general-purpose MCP servers, it incorporates detailed knowledge of penetration testing strategies, tool logic, and CTF methodologies, enabling nuanced and contextually relevant recommendations. Its reasoning engine adapts to the current system state, leveraging resources such as CVE databases, port-service mappings, and OS-specific exploits to prioritize attack steps and suggest optimal paths. The server integrates both Beam Search, for efficient linear planning, and Monte

Carlo Tree Search (MCTS), for exploratory planning, offering comprehensive coverage of potential attack chains. Additionally, it is tool-aware, recommending specific utilities like **rustscan**, **nmap**, **enum4linux**, or **linpeas**, complete with rationales for their selection. The outputs are structured and detailed, including complete attack chains, node scores, critical path indicators, and relevant statistics. The server's LLM-ready API allows seamless integration into platforms like Claude, Cursor, and LangChain agents, positioning it as a versatile and robust reasoning engine for automated penetration testing systems.

2) Setup and Integration of Our Server: Our MCP server is available at publicly accessible repository [11]. After cloning the repository, run *npm install && npm run build* to install dependencies and generate the ready-to-use `index.js` entry point.

The next step is to register the MCP server with the client platform. Open the configuration interface of the chosen agent platform (e.g., Claude, Cursor, LangChain) and specify the path to the built `index.js` file. Additionally, set the tool name in the platform configuration to `pentestthinkingMCP` to ensure that the client can recognize and communicate with the server.

Following registration, we establish a connection to the execution environment. The MCP client should be connected to an operating system (e.g., a Linux VM) that is equipped with standard tools commonly used in penetration testing, such as Rustscan, Nmap, Metasploit, Ghidra, and other utilities. Ensure that the client can relay system state information (e.g., open ports, service banners) to the MCP server and receive structured outputs in response.

Next, we enable feedback loops by configuring the MCP client to feed the results of executed actions (e.g., terminal output from the Linux VM) back into the MCP server. This feedback loop allows the server's reasoning engine to dynamically adjust its strategy, prioritize subsequent steps, and retain contextual awareness across multiple interactions.

We then integrate additional MCP modules as needed. For example, `FilesystemMCP` for file system operations, `BrowserMCP` for automatic web interface interactions, or `WebSearchMCP` for external data retrieval and CVE lookups. Each module's path and tool name must be properly configured in the client to extend the system's capabilities.

Finally, we test and verify that the MCP server provides coherent reasoning outputs (e.g., tool suggestions and next-step recommendations), that the client correctly executes the actions, and that logs confirm smooth coordination among the MCP client, the execution environment, and the server.

3) Overall flow: The PentestThinkingMCP Server, when integrated into our architecture, takes natural language prompts that convey the attacker's goal, target system, available tools, and operating limitations. A typical prompt may delineate the target IP (e.g., 10.10.10.3), the overarching objective (e.g., extract the user flag), and stipulations such as operating autonomously, minimizing latency, and avoiding unnecessary privilege escalation. The server dynamically analyzes known vulnerabilities (e.g., CVEs), port-service mappings, and system state (e.g., open ports) to generate organized, step-by-step suggestions. The outputs encompass specific instructions (e.g., "Run nmap -p- 10.10.10.3"), rationales, path prioritization, and score criteria that direct the LLM or human operator. This prompt-based interface facilitates quick, context-sensitive decision-making customized for practical penetration testing processes.

D. Evaluation Strategy

To evaluate the effectiveness of the LLM-assisted pipeline, we compare its performance against manual attempts by human testers of differing skill levels (described in the Experiments section). We use HackTheBox virtual machines as testbeds, all categorized as "Easy," but varying in terms of vulnerability types, services, and initial access vectors. We also spawn a dedicated virtual machine (VM) with support from the company Ethical Byte and set it up to obtain initial access to it, enabling the LLM to operate in a realistic, tool-equipped environment with the option of receiving minimal online assistance when needed (e.g., to execute commands or inspect system state). Performance is measured in terms of time taken, number of steps, and success in achieving initial access.

IV. Experiments

A. Experimental Setup

To assess the efficacy of our software (LIMA) during the initial access phase, we devised a controlled experiment utilizing VMs from the HackTheBox platform [14]. Our aim was to evaluate the efficacy of human testers with varying skill levels in comparison to an LLM-assisted system for gaining first access to target computers.

We selected four HackTheBox machines, namely, LAME, PC, TwoMillion, and Support, and spawned a Linux VM "Exam Box" as test cases. These boxes are categorized as "Easy" in terms of overall difficulty but exhibit diversity in terms of vulnerability types, exploited services, attack categories, and required exploitation techniques. This ensured that while the machines were accessible to beginners, they still offered enough variation to evaluate generalization across attack vectors.

Each machine was evaluated under two conditions:

- A human tester (either beginner or intermediate) attempted to manually gain initial access using standard penetration testing methodology.
- The same machine was then tested using LIMA designed to replicate the decision-making and tool usage of a human tester.

To ensure a fair comparison, both human testers and LIMA were given access to the same preliminary information: service banners, `rustscan` and `nmap` output, and any publicly available data relevant to the target system. All experiments were conducted in a sandbox environment to ensure security and reproducibility.

Key characteristics of the selected machines, including their primary vulnerabilities and exploitation categories, are

TABLE I
SUMMARY OF CATEGORIES AND VULNERABILITIES OF MACHINES USED IN OUR EXPERIMENTS

Box Name	Category	Tools Used	Initial Access Method
LAME	1. Vulnerability Assessment 2. Enterprise Network	1. Nmap 2. Metasploit	Remote Code Execution
TwoMillion	1. Web Application	1. Burp Suite 2. Cyber Chef	Command Injection
Support	1. Vulnerability Assessment 2. Enterprise Network 3. Security Operations 4. Web Application	1. DNSpy 2. Wireshark 3. BloodHound 4. LDAPSearch	Plain-text Credentials
PC	1. Vulnerability Assessment 2. Enterprise Network 3. Web Application	1. grpcurl	SQL Injection
Exam Box	1. Web Application 2. Enumeration	1. Nmap 2. ROT13 3. Hashcat 4. CyberChef	SSH Login

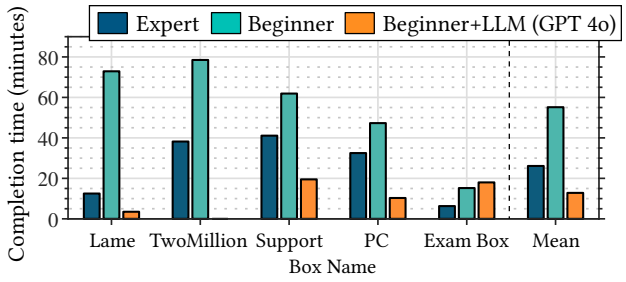

Fig. 3. Completion time for initial access on the target machines. LIMA substantially reduces the time needed to achieve initial access. LIMA failed to gain initial access on the TwoMillion, because the exploitation path required BurpSuite functionality which is not yet available in our MCP stack.

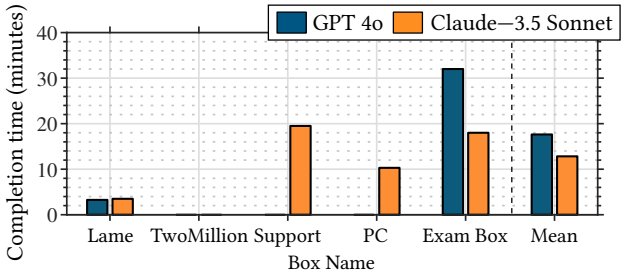

Fig. 4. LIMA-only completion times for initial access on five targets.

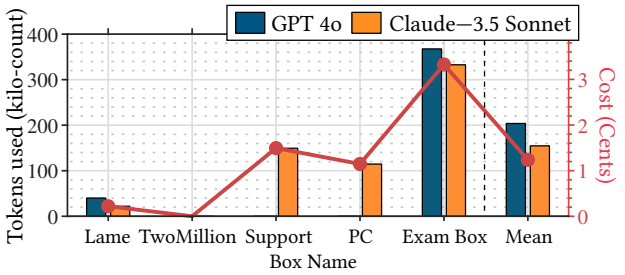

Fig. 5. Token consumption and API cost from LIMA runs. Overall, Claude incurs higher token usage and cost per box, whereas GPT-4o is more economical except on Exam Box, where both models required similar resources.

summarized in Table I. Subsequent tables present the methods used for initial access and the time taken by each participant.

B. Manual Access Baseline

To establish a baseline for evaluating LLM-assisted penetration testing, we first conducted a manual assessment of the selected HackTheBox (HTB) machines and Exam Box. Three human testers participated in this study:

- **Tester A (Expert):** Four years of Capture The Flag (CTF) experience and an expert-level penetration testing skills.
- **Tester B (Beginner):** Basic knowledge of penetration testing with limited hands-on experience.
- **Tester C (Beginner):** Basic knowledge of penetration testing with limited hands-on experience, but receives assistance from LIMA.

The testers followed a standard penetration testing methodology to gain initial access to each machine. This included reconnaissance, scanning (with service enumeration), vulnerability identification, and exploitation. The target machines were classified as "Easy" by the HTB platform, but they were distinct in terms of services, vulnerabilities, and exploitation paths.

Despite the shared methodology, Tester A and Tester B, demonstrated significant differences in performance:

- **Time to Access:** The beginner tester (Tester B) took considerably longer across all machines due to unfamiliarity with key services and the need to research tools, syntax, and exploitation strategies.
- **Service Understanding:** Tester B often had to learn the function and significance of the services discovered (e.g., SMB, FTP, or Apache), while Tester A was able to leverage prior knowledge to navigate more efficiently.
- **Enumeration Strategy:** Tester A performed more focused and effective enumeration, quickly identifying misconfigurations or outdated software, whereas Tester B approached this step more broadly.
- **Exploit Generation and Use:** Exploiting the identified vulnerabilities was the most time-consuming step for the beginner, particularly when manual adaptation of exploit code was required. In contrast, Tester A was able to find or adapt exploits with greater ease. This phase is also considered one of the most complex in the penetration testing process [3].

Completion time varied dramatically across testers: the beginner (Tester B) averaged more than one hour per box, whereas the LIMA-assisted beginner finished in under five minutes and even outpaced the expert on two machines (Figure 3).

LIMA performed exceptionally well on the machines LAME, Support, PC, and Exam Box. When we removed the human from the loop, GPT-4o completed one target (LAME) in under five minutes, whereas Claude completed two targets (LAME and Exam Box) and required up to 20 minutes. Claude completed PC with human assistance (Tester C). Both GPT-4o and Claude failed on TwoMillion (Figure 4) because that box's exploit path requires BurpSuite functionality, which is not yet available in our MCP stack. Among the two GPT models tested, Claude 3.5 demonstrated greater autonomy, independently executing most steps. Claude was able to run Linux commands, install necessary software, browse the web, and complete tasks such as creating and registering users using invitation codes via BrowserMCP, all without human assistance. However, it required manual input to solve CAPTCHAs. In contrast, ChatGPT-4o needed human support at several points, including software installation and interpreting command outputs. It also failed to extract critical information from a `UserInfo.exe` file, which was essential for gaining initial access to the Support machine from a Linux environment.

Token consumption differed sharply between the two models, with Claude averaging almost twice as many tokens per box and therefore incurring a higher API cost (Figure 5).

Here are some key observations.

1) *Pattern-matching strength*: Both LLMs quickly mapped SMB and FTP banners on `Lame` to known CVEs (e.g. CVE-2007-2447), a step that took the beginner tester several minutes.
2) *Adaptive weakness*: When the first payload for `Support` failed, Claude repeated variants of the same exploit three times, whereas the expert tester switched to a different enumeration path (SMTP misconfig).
3) *Non-standard configs*: GPT-4o could not parse the custom web login on `PC`, highlighting limits on UI-driven targets unless BrowserMCP is improved.
4) *CAPTCHA bottleneck*: Both models stalled on the CAPTCHA gate of the exam VM until a human entered the challenge, after which Claude finished in 90 s.

These examples reinforce our quantitative baseline: LLMs excel at knowledge retrieval and straightforward exploit mapping but still rely on human creativity for atypical configurations and interactive hurdles.

V. Conclusion

This paper introduced LIMA (Large-language-model-driven Initial Machine Access), a modular framework that combines off-the-shelf large language models with Model Context Protocol (MCP) servers to automate initial access. We benchmarked Claude 3.5 and GPT-4o on four public HackTheBox machines and one private VM, alongside beginner- and expert-level human testers. Claude autonomously captured user-level flags on three of five targets, needing help only for CAPTCHA prompts; GPT-4o achieved similar coverage but required more frequent guidance. The lone failure—TwoMillion—was due to the absence of a BurpSuite MCP module, highlighting the importance of context-specific tooling. These results provide the first quantitative baseline for AI-driven initial access on live, heterogeneous systems. Future work will extend LIMA with additional MCP modules, full GUI automation, and systematic evaluation of defensive counter-measures. In the near term, offensive tool builders should prioritise MCP support for GUI-dependent utilities (e.g., BurpSuite) and CAPTCHA solvers, while defenders can raise the bar by inserting interactive checkpoints and monitoring for LLM-style reconnaissance patterns.

References

[1] EC-Council. Learn about the five penetration testing phases, 2022. Accessed: May 16, 2025.

[2] Tarun Yadav and Arvind Mallari Rao. Technical aspects of cyber kill chain. In Jemal H. Abawajy, Sougata Mukherjea, Sabu M. Thampi, and Antonio Ruiz-Martínez, editors, *Security in Computing and Communications*, pages 438–452, 2015.

[3] Thanassis Avgerinos, Sang Kil Cha, Alexandre Rebert, Edward J. Schwartz, Maverick Woo, and David Brumley. Automatic exploit generation. *Communications of the ACM*, 57(2):74–84, February 2014.

[4] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. Pentestgpt: evaluating and harnessing large language models for automated penetration testing. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, 2024.

[5] Andreas Happe and Jürgen 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 2023, page 2082–2086, 2023.

[6] Charilaos Skandylas and Mikael Asplund. Automated penetration testing: Formalization and realization. *Computers & Security*, 155:104454, 2025.

[7] Xiaofeng Zhong, Yunlong Zhang, and Jingju Liu. Penqa: A comprehensive instructional dataset for enhancing penetration testing capabilities in language models. *Applied Sciences*, 15(4), 2025.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[9] Andreas Happe and Jürgen 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 2023, page 1669–1680, 2023.

[10] Filesystem Model Context Protocol. Filesystem mcp server: Node.js server implementing model context protocol (mcp) for filesystem operations. https://github.com/modelcontextprotocol/servers/tree/main/src/filesystem, 2025. Accessed: 2025-07-12.

[11] M. I. Saleem. Pentestthinkingmcp: A modular reasoning engine for ai-assisted penetration testing. https://github.com/ibrahimsaleem/PentestThinkingMCP, 2025. Accessed: 2025-07-12.

[12] MCP Browser. Mcp browser: Connect ai apps to your browser to automate tests and tasks. https://browsermcp.io/. Accessed: 2025-07-12.

[13] Exa AI. Web search for llms: One api to get any information from the web, built for ai products. https://exa.ai/. Accessed: 2025-07-12.

[14] Hack The Box Ltd. https://www.hackthebox.com/.