PenTest2.0: Advancing Ethical Hacking with GenAI–Driven Privilege Escalation

Haitham S. Al-Sinani[1,3], Chris J. Mitchell[2], and Abdulaziz S. Al-Hosni[1]

[1] Diwan of Royal Court, Muscat, Oman. [hsssinani,ashosni]@diwan.gov.om
[2] Royal Holloway, University of London, Egham, UK. C.Mitchell@rhul.ac.uk
[3] German University of Technology in Oman, Muscat, Oman.
Haitham.AlSinani@gutech.edu.om

Abstract. Ethical hacking remains time-consuming, difficult to scale, and prone to human error because it depends on experts manually executing complex command sequences. `PenTest++`, an AI-augmented system supporting key penetration-testing workflows, was proposed to address this, but it lacks privilege escalation (PrivEsc). We present here `PenTest2.0`, a major evolution enabling multi-turn, automated privilege escalation powered by Large Language Model reasoning. It adds Retrieval-Augmented Generation, Chain-of-Thought prompting, PenTest Task Trees, and optional human hints to improve reasoning, context retention, and adaptability. We describe its design and a proof-of-concept implementation, and show it can perform adaptive privilege escalation. We also highlight limitations related to prompt sensitivity, execution context, and semantic drift. `PenTest2.0` offers a practical step toward scalable AI-driven penetration testing, although further research is needed to ensure reliable and safe behaviour in security-critical settings.

Keywords: AI · Ethical Hacking · PrivEsc · GenAI · ChatGPT · LLM

1 Introduction

Ethical hacking, or penetration testing [16] (PenTesting), is labour-intensive, error-prone, and hard to scale because it relies on experts manually crafting commands, interpreting outputs, and adapting strategies. Our earlier `PenTest++` [4] reduced this burden by integrating GenAI into reconnaissance, scanning, enumeration, exploitation, and reporting, but lacked automated Privilege Escalation (PrivEsc) — a crucial post-exploitation step — and did not support Retrieval-Augmented Generation (RAG), Chain-of-Thought (CoT) prompting, PenTest Task Trees (PTTs), or human-injected hints. To overcome these shortcomings, we propose `PenTest2.0`, that performs multi-turn, GenAI-driven PrivEsc and optionally incorporates RAG, CoT, PTTs, and human hints, while logging commands, reasoning traces, cost metrics, and diagnostics for detailed analysis.

CoT prompting [18] encourages the LLM to express intermediate reasoning before producing an action. *RAG* grounds LLM outputs by injecting concise, context-relevant facts from external sources, thereby reducing hallucination and aligning suggestions with practised techniques [14]. Originally adopted in PenTestGPT [9], the *PTT* preserves task data across turns, addressing context loss.

We address these research questions (RQ). **RQ1:** How effectively can GenAI autonomously accomplish PrivEsc with human supervision? **RQ2:** To what extent do RAG, CoT, PTTs, and human hints improve the effectiveness and traceability of GenAI-driven PrivEsc? **RQ3:** What practical limitations arise when applying LLMs to command-based PrivEsc tasks in realistic environments? We make five contributions: **C1:** demonstrating the feasibility of automating PrivEsc with user oversight; **C2:** a GenAI-powered, multi-turn PrivEsc prototype able to reason, generate, execute, and adapt commands, released as open-source[4]; **C3:** integrating optional RAG, CoT prompting, PTTs, and human hints for deeper reasoning; **C4:** comprehensive evaluation on a realistic Linux target, capturing autonomous behaviour and failure modes; and **C5:** a critical review of strengths, limitations, and future research directions for GenAI-assisted PenTesting.

The paper is organised as follows. Sections 2 and 3 describe the system's operation and design rationale, respectively. Section 4 outlines the prototype, while Sections 5 and 6 present an evaluation and cost analysis. Section 7 discusses broader implications, and Section 8 reviews related research. Finally, Section 9 concludes the paper and presents future work directions.

2 PenTest2.0 Operation & Design Choices

Since `PenTest2.0` extends `PenTest++` [4], we summarise the `PenTest++` workflow showing where `PenTest2.0` integrates. After automated **reconnaissance** to discover live hosts and gather preliminary network and service information, `PenTest++` performs scanning and **enumeration** (e.g., using `nmap` and `gobuster`) to identify ports, services, and configuration details relevant to attack planning. During **exploitation**, it proposes tailored payloads and command sequences targeting discovered vulnerabilities, with user approval throughout. Once a foothold is achieved, `PenTest2.0` takes over **post-exploitation** to handle PrivEsc. Finally, `PenTest++` compiles logs, findings, and recommendations into **structured reports**, using GenAI to enhance clarity and readability. We now describe how `PenTest2.0` operates (see Figs. 1 and 2).

1. **Prior Assumption:** `PenTest2.0` presumes prior access to the target system, usually via a low-privileged shell from a preceding exploitation stage. This corresponds to standard post-exploitation settings in PenTesting research, and focuses the system's scope on PrivEsc.
2. **System Context Collection via Probing:** Prior to calling the LLM, our system automatically runs a predefined set of reconnaissance commands on the target to obtain information about the system environment, runtime state, and operating context. Typical commands include `id`, `whoami`, `hostname`, `uname -a`, `sudo -l`, `env`, and `ls -la /tmp`. Their outputs

[4] `https://github.com/DrHaitham/PenTest2.0`

are aggregated, condensed, and used to form the initial GenAI prompt, allowing the LLM to start reasoning from a precise and cost-aware snapshot of the target.

3. **Prompt Generation:** Based on the collected system state, `PenTest2.0` generates the initial prompt by inserting relevant details into a predefined template. The resulting prompt is then sent to the LLM to start the first reasoning turn, supplying the context required to propose an appropriate PrivEsc strategy.
4. **Execution–Feedback Cycle:** After receiving a command suggestion from the LLM, `PenTest2.0` runs it on the target host through SSH and records the resulting output. If root access is obtained, the procedure stops. Otherwise, `PenTest2.0` prepares a new prompt by combining a condensed form of the previous prompt with the most recent output and submits it to the LLM for the next reasoning step. This cycle repeats until either root privileges are gained or the predefined turn limit is reached.
5. **User Oversight:** While the system is designed to support full automated operation, `PenTest2.0` requires user supervision at critical checkpoints. Before any prompt is submitted to the LLM, the user inspects the complete prompt content, token count, and estimated API cost. In the same way, every command proposed by the LLM must receive explicit user approval prior to execution on the target system. This two-step approval approach promotes responsible and safe use, reduces the risk of unintended system impact or unnecessary expenditure, and ensures that the user retains final authority over all actions.
6. **Optional Extensions:** `PenTest2.0` provides a set of optional mechanisms intended to improve reasoning quality and reduce the time required to obtain root access: **RAG** — available in offline and online forms — to supply relevant knowledge to LLM prompts; **CoT Prompting** to support stepwise intermediate reasoning; **PTTs** to maintain persistent records of subtasks, executed commands, and observations across turns; and **Human Hints** to steer the LLM away from ineffective behaviours or towards more promising strategies (see Fig. 2). These optional components can be toggled on or off as needed; if activated, they are incorporated dynamically during execution.
7. **Runtime Logging and Reporting:** Throughout execution, `PenTest2.0` records each prompt, issued command, resulting output, token consumption, and root verification step. At the end of a run — either after success or upon reaching the turn limit — the system produces structured reports summarising decisions, costs, and results, enabling reproducibility, traceability, and effective debugging.

3 Design Choices

We now describe the rationale behind the design choices of `PenTest2.0`.

Modular Architecture: `PenTest2.0` adopts a modular, component-based architecture, facilitating extensibility, debugging, and customisation. Core modules such as `command_executor.py`, `llm_connector.py`, `ptt_manager.py` and `shell_root_detection.py` encapsulate distinct responsibilities. This *separation of concerns* ensures that the system can be maintained, extended, or debugged without introducing regressions elsewhere.

Dynamic Prompting: `PenTest2.0` builds LLM prompts dynamically from small, reusable blocks (system facts, recent outputs, task history, and optional modules) rather than a single monolithic prompt. At runtime the user can toggle CoT, RAG, PTT and human hints (e.g. via CLI flags), enabling the system to emit either a minimal, cost-efficient prompt containing only essential context or a richer prompt that embeds reasoning chains and retrieved knowledge when deeper analysis is needed. This composable approach permits precise prompt tailoring without restarting the system, reduces token usage in routine runs, and allows rapid escalation to intelligence-heavy modes if needed.

The base prompt in `PenTest2.0` defines the operating context, rules of engagement, and input/output structure for the LLM.

```
Base prompt (excerpt): You assist with Linux privilege escalation. From a
low-privileged account (USERNAME), propose a safe next command each turn, using
the system summary and output, to reach root within MAX_TURNS ...
Rules: (1) never repeat a successful command; (2) retry corrected versions of
syntax-error failures; (3) avoid destructive commands (e.g. rm -rf *) ...
JSON Output:

{  "command\_non\_interactive": "string, for automated execution (no \$, \#, `)",
  "command\_interactive": "string, interactive version if possible, else empty",
  "system\_summary": "string, max 10 very short bullet points",
  "command\_history": "string (max 15 lines, summarised cleanly)",
  "rationale": "string, 1{2 sentences explaining why this command was chosen" }
```

The system incrementally rebuilds the prompt at each turn by injecting a concise summary of the current system state and recent outputs, preserving essential context while bounding token usage. This turn-level, iterative injection lets the LLM refine strategy from real execution feedback, avoids prompt bloat, and yields more reliable decision-making than single-shot prompts.

Enforced Reasoning: `PenTest2.0` requires the LLM to output a brief rationale with every command, encouraging reflective reasoning rather than shallow pattern matching. By explaining why a command fits the current system state and prior outputs, the model reduces repeated failures, exposes its logic for user inspection, and leaves a lightweight audit trail. This improves decision quality, helps detect drift or hallucinations, and simplifies debugging, making rationale generation a compact but essential mechanism for reliable PrivEsc.

CoT prompting: Instead of proposing an action directly, the LLM is instructed to examine the environment summary, reflect on the previous command and its output, and then determine the next step. This supports more deliberate behaviour, reduces unnecessary repetition, and improves transparency during multi-turn PrivEsc attempts. We implement CoT by adding a short directive to the prompt that encourages step-wise reasoning. We use two lightweight modes: "`zero-shot CoT`", where only this instruction is added: *'Think step by step. First, assess the system summary for PrivEsc paths. Then, evaluate the last command and output. Finally, decide on the most logical next command'*;

and "`few-shot CoT`", where concise examples, drawn from earlier successful PrivEsc runs, show the expected reasoning and output structure.

Hints: `PenTest2.0` employs **human hints**, which allow a pen-tester to inject concise, structured guidance into the LLM prompt, leveraging operator knowledge to steer reasoning and reduce wasted turns. For example: 'Human Hint: *use the 'id' command instead of the '/bin/sh' for automated root verification*'. This lightweight human-in-the-loop (HITL) [8] mechanism improves convergence and control while adding negligible prompt overhead.

Non-Interactive Commands: To prevent shell hangs caused by LLM-suggested interactive commands (e.g., `sudo su`), `PenTest2.0` requires the model to output both a safe non-interactive command for automated SSH execution and an optional interactive variant for manual use. This dual-format instruction improves reliability, avoids blocking, and preserves operator control during multi-turn PrivEsc.

RAG: `PenTest2.0` employs a hybrid, low-overhead RAG design: *offline*, a curated corpus (e.g. GTFOBins[5]) is pre-downloaded, indexed, and queried locally to supply short, targeted snippets without incurring runtime latency or network dependence; and *online*, lightweight live lookups that retrieve up-to-date snippets which are minimally summarised and injected into the prompt only when the LLM requests external support. In either case, retrieved snippets are inserted in a compact, structured form so the LLM can use them to justify or refine a non-interactive command. This pragmatic RAG approach balances relevance, prompt-size control, and operational reliability.

PTT: `PenTest2.0` extends the PTT by allowing dynamic updates of 'updated_statuses', 'new_subtasks', and 'commands_to_avoid'. A sample is as follows. "Current PTT Status: **Subtask 1**: *Examine sudo privileges. Status: pending.* **Subtask 2**: *Identify potential misconfigurations in awk. Status: pending.*"

Prompt Cost Control: `PenTest2.0` computes token count and estimated API cost before every LLM call and displays them to the user for approval. This prevents prompt bloat, avoids unexpected charges, and enforces human oversight in multi-turn runs; an approach informed by early tests where unchecked prompt growth rapidly consumed credits. The approval step is repeated each turn to maintain strict cost and governance control.

Safe Command Execution: Experiments show that LLM suggestions may be misleading or even hazardous. The LLM might, for instance, generate unsafe commands such as `rm -rfv *`, which could delete the entire filesystem, or computationally expensive actions like `zip -rv zipped.zip /`, which attempts to compress the full system recursively. To prevent this, `PenTest2.0` combines a local blacklist of unsafe patterns with mandatory human approval. Any command matching the blacklist is discarded automatically, while all remaining suggestions are shown to the user, together with the LLM's rationale, for explicit confirmation before execution. Only authorised commands are executed via SSH, with outputs logged for later reasoning and reporting. This dual-approval

[5] `https://gtfobins.github.io/`

process, consisting of static command filtering and real-time human validation, ensures both operational safety and ethical control.

LLM Safety Mechanisms: Some LLMs employ built-in safety and policy enforcement mechanisms that may cause them to refuse or partially redact responses to prompts related to sensitive security scenarios, including PenTesting and exploit development. Although this behaviour was not observed in our PoC experiments, it represents a practical consideration for GenAI-assisted security tooling and can often be mitigated through model substitution, as different LLMs exhibit varying degrees of strictness and policy interpretation. Recent studies [10, 17, 20] further indicate that such safeguards are not absolute and may be circumvented under certain conditions via techniques such as prompt re-framing, multi-turn context manipulation, and role-based conditioning, highlighting that safety enforcement in contemporary LLMs remains probabilistic and model-dependent rather than formally guaranteed.

4 Prototype Implementation

`PenTest2.0` is implemented in Python 3 for its flexibility and strong library ecosystem. All experiments were conducted in VirtualBox 7 running on a physical host, a Lenovo Windows 11 laptop (Intel Ultra 7, 32 GB RAM). The virtual environment consisted of a Kali Linux VM running `PenTest2.0` and a lightweight Debian-based target VM, connected via a NAT Network to simulate an isolated test environment. For GenAI integration, `PenTest2.0` uses OpenAI's `gpt-4o-mini` model through the official API, selected for its favourable cost–performance trade-off. While more advanced models (e.g., `o3`, `gpt-4.1`, `gpt-5`) are supported, they were not used due to higher token costs. Alternative LLM providers (e.g., Gemini, DeepSeek) remain possible options. `PenTest2.0` functions as an AI-driven, modular, multi-turn, PrivEsc agent, as outlined below.

1. **Initial Setup:** The prototype is a modular Python3 application running on a Kali VM and assumes an existing low-privilege foothold on the target. It is launched via CLI with a config file (target IP, SSH credentials, LLM model, max turns) and optional flags (CoT, RAG, PTT, hints). On startup it verifies Internet and SSH connectivity, initialises core modules (e.g. `command_executor`, `llm_connector`, `shell_root_detection`, etc.), creates timestamped logs and counters, and loads optional knowledge stores.
2. **Bootstrapping and Reconnaissance:** On launch, it initialises configured modules (e.g. RAG, CoT, PTT, human hints), performs a compact SSH scan of the target (e.g. `whoami`, `id`, `hostname`, `uname -a`, `cat /etc/os-release`, `sudo -l`, `ss -tulnp`, `df -h`, `free -m`, `ps aux --sort=-%mem | head -n 10`, `ls -la /tmp`, `find / -perm -4000 -type f 2>/dev/null`), parses and summarises outputs.
3. **Prompt Construction:** It assembles a structured LLM prompt from gathered reconnaissance using Python template logic; optional components (CoT, RAG, PTT, human hints) are injected only if enabled. The prompt contains: (a) **system summary:** concise OS, user, and privilege facts (from

`id`, `uname -a`, `sudo -l`, `env`, etc.); (b) **command history:** a recent, size-capped list of attempted commands and outputs (oldest entries dropped to avoid prompt bloat); (c) **reasoning instructions:** a short directive (e.g. 'think step by step') and optional CoT scaffolding; (d) **task goals/response schema:** require a single, valid JSON object containing at minimum a *non_interactive* command, an optional *interactive* variant, a brief *system_summary*, a short *command_history*, and a two-sentence *rationale*; and (e) **optional enhancements:** conditionally add RAG excerpts, PTT state, or a human hint when requested. To ensure parsing and automation, the LLM is instructed to return one compact JSON object only.

4. **Prompt Submission:** `PenTest2.0` first computes the token count of the prompt, estimates its cost for the chosen model, and displays this to the user. The system submits the prompt to the LLM only after explicit user approval, preventing oversized prompts and unintended API expenditure.
5. **LLM Response:** Upon receiving the prompt, the LLM returns a single compact JSON object containing the following fields: 'non_interactive_command' (for automated root detection); `interactive_command` (manual variant); 'system_summary' (concise key facts); `command_history` (a summarised, capped list of previously executed commands and their outputs); and 'rationale' (a brief, context-aware justification explaining why the proposed command is suitable). If PTT is enabled, the object may include `ptt_update` with `initial_tree`, `new_subtasks`, `updated_statuses`, & `commands`.
6. **User Approval:** `PenTest2.0` presents the proposed command and its concise rationale to the operator for explicit approval; only user-approved commands are executed via SSH and their outputs logged for next turns.
7. **Iterative Feedback Process:** If root privileges are not acquired (e.g., when the output lacks `uid=0(root)`), `PenTest2.0` condenses the command output and prepares the next prompt. This prompt integrates the updated system context, command history, and any relevant PTT changes. The procedure continues for a fixed number of turns or until PrivEsc succeeds. In later turns, both LLM requests and responses follow the structured format specified in steps 3 and 5, ensuring consistent and traceable interactions.
8. **Optional enhancements:** At runtime, users may enable: (i) local RAG from a FAISS-backed markdown store (FAISS: Facebook AI Similarity Search for fast vector lookup, e.g. GTFOBins); (ii) online GTFOBins retrieval; (iii) CoT reasoning; (iv) human-injected hints; and, (v) PTT tracking, which tags commands with task IDs and updates task statuses (*pending*, *in_progress*, *done*, *skipped*) across turns.
9. **Root Detection and Termination:** After each execution, `PenTest2.0` applies regex-based checks to detect root access; if confirmed, the loop terminates immediately, otherwise the next turn begins.
10. **Logging and Reporting:** The system records all prompts, commands, outputs, reasoning traces, token metrics, and PTT updates, and produces structured logs and a final session summary, with optional visualisations.

5 Test Results

We evaluated `PenTest2.0` across seven system configurations combining or excluding CoT, hints, RAG, and PTT. All tests targeted a vulnerable Linux VM with known PrivEsc vectors to assess robustness, automation performance, and feature-specific behaviour. Table 1 summarises the results, where all seven configurations attained root (either automatically or with manual confirmation). Four yielded automatic root detection, and the other three failed to detect root because the LLM-proposed commands (such as: `sudo awk 'BEGIN {system("/bin/sh")}'`) spawn an interactive shell that the non-interactive SSH wrapper cannot observe; we plan to investigate this. Configurations 1–4 used non-interactive variants (e.g. `sudo awk 'BEGIN {system("id")}'`), allowing the ShellDetector to parse outputs such as `uid=0(root)` and terminate the loop early. Table 2 gives an overview of configuration performance across key dimensions, showing trade-offs in speed, cost, accuracy, and feature richness.

Table 1: PenTest2.0 results across seven configurations (max. 10 turns).

Config	**–CoT**	**–Hint**	**–CoT –Hint**	**ALL**	**–RAG**	**–PTT**	**None**
Root	✓	✓	✓	✓	✓	✓	✓
Auto-root	✓	✓	✓	✓	Root manually confirmed		
Turns	1	2	2	2	10	10	10

Table 2: Compressed summary of PenTest2.0 configuration characteristics.

Fastest to Root	**Cost-Effective**	**Best Balance**	**Feature-Rich**
`CoT Hint`	`CoT`	`CoT Hint`	`CoT Hint RAG PTT`
Turn 1	Lowest tokens	Early root	All flags; highest cost

6 Cost Analysis

`PenTest2.0` performs cost tracking using OpenAI's July 2025 pricing. For each turn, we recorded prompt and completion tokens and applied the model's rates, e.g. \$0.15/M prompt tokens and \$0.60/M completion tokens. Total session cost is the sum of per-turn input and output charges. When needed, word counts are estimated using 1 word $\approx$ 1.33 tokens, and completion size is approximated as 40% of the prompt. For example, a 5,000-token prompt with a 2,000-token completion costs \$0.00195 per turn. 7 configurations were evaluated, each permitted up to 10 turns. Results appear in Table 3. Early-success setups (*CoT+Hint*, *HumanHint*, *CoT*) achieved root in 1–2 turns at very low cost, whereas verbose or reasoning-heavy configurations (*No-Flags*, *RAG*, *PTT*) reached the 10-turn cap without auto-detecting root access, consuming more tokens but offering no accuracy advantage, showing that higher-cost configs do not necessarily yield better performance, as excessive verbosity can degrade effectiveness and inflate cost. Lightweight prompting (*CoT+Hint* or *Hint*) offers the best speed and cost.

Table 3: Tabulated cost (in $) and turn count across configurations.

Config.	CoT+Hint	Hint	CoT	ALL	NoFlags	RAG	PTT
Turns	1	2	2	2	10	10	10
Cost	0.00053	0.00066	0.00091	0.00207	0.00310	0.00394	0.00623

7 General Discussion

Research Questions: For **RQ1**, our experiments show that LLMs, e.g. GPT-4o, are capable of autonomously discovering and carrying out PrivEsc techniques on realistic Linux systems, often achieving root within 1–2 turns in configurations like *CoT* or *Hint*. However, because LLMs may propose unsafe or interactive commands, non-interactive enforcement and HITL supervision remain essential for safe and successful operation. For **RQ2**, CoT and human hints consistently improved reasoning depth, reduced repetition, and accelerated escalation, while RAG and PTT enhanced contextual grounding and traceability. Yet these heavier techniques also increased prompt size and cost, and often did not improve performance due to prompt stress or hallucinations. The *–CoT –Hint* configuration offered the best balance of efficiency, reliability, and cost-effectiveness. For **RQ3**, several limitations emerged: LLMs occasionally hallucinated unsafe or invalid commands, ignored output rules, repeated failing strategies, or produced interactive shells that bypassed automated root detection. Some commands were resource-intensive enough to risk system instability. These issues highlight the need for controlled prompt design, command filtering, and HITL oversight.

Benefits: `PenTest2.0` extends its predecessor by introducing autonomous, multi-turn PrivEsc powered by GenAI while preserving safety and user control. It automates escalation through an iterative reasoning loop that adapts to command outputs, supports advanced techniques such as CoT, RAG, PTTs, and optional human hints, and maintains structured execution with robust logging for auditability. Lightweight human collaboration improves success rates without compromising automation, and a modular Python architecture enables extensibility and open-source adoption. Safety and cost control are enforced through token-cost estimation, command blacklisting, and mandatory user approval, ensuring responsible and efficient operation in security-critical contexts.

Limitations: There remain limits to `PenTest2.0` capabilities. The LLM may still produce ineffective, unsafe, or ambiguous commands, particularly in hardened or unfamiliar environments, and automated execution — although mitigated by blacklisting, user approval, and cost checks — carries inherent operational risk. Reliance on cloud-hosted LLMs raises privacy and compliance concerns, requiring careful adherence to organisational policies. Moreover, all evaluations were conducted on controlled Linux targets with known PrivEsc vectors, limiting generalisability. In addition, intrinsic LLM shortcomings such as hallucinations, prompt sensitivity, and semantic drift persist, reinforcing the need for continued HITL oversight to ensure correctness and safe use in offensive security contexts. Finally, our experimentation show that the LLM also exhibited unstable behaviours, including repeatedly proposing the same ineffective command and occasionally generating resource-intensive actions capable of degrading target VMs. While stronger models may reduce such failures, they do so at a substantially higher cost, underscoring a reliability–affordability trade-off.

8 Related Work

GenAI is increasingly used in cybersecurity, spanning both defensive and offensive security, including ethical hacking. Despite significant progress [1, 9, 11–13, 15, 19], a fully autonomous, comprehensive PenTesting system remains elusive. Many such systems have emerged in parallel with our work, which included proposing conceptual frameworks for GenAI-driven PenTesting [6], evaluating LLM performance in Windows [2] and Linux environments [7], and analysed GenAI-supported exploitation and manual PrivEsc [3]. We showed that GenAI can accelerate decision-making and automate parts of the PenTesting workflow.

PentestGPT [9] is a recent LLM-based assistant that employs reasoning, generation, and parsing modules, together with PTTs, to guide users through manual command execution in a HITL workflow. However, its reliance on manual user execution limits automation. To improve upon this, `PenTest++` [4] automated predefined PenTesting actions, governed by user approval.

9 Conclusions and Future Work

This paper introduced `PenTest2.0`, an extension of `PenTest++` that concentrates on automating the PrivEsc phase of PenTesting. The system was built as a GenAI-enhanced prototype that operates across multiple interaction rounds, generates and executes commands in real-time, and updates its behaviour using feedback from the target. Experimental tests carried out under several configuration settings, including CoT, RAG, PTT, and operator hints, show that GenAI support can be applied to PrivEsc in a predictable and manageable manner when combined with explicit task control and human supervision. The results further suggest that guided reasoning with hints offers a reasonable trade-off between execution time, stability, and monetary cost. To reduce risks associated with LLM failures, `PenTest2.0` incorporates structured task tracking, token-aware prompt construction, and built-in safety mechanisms. Nevertheless, practical limitations were observed, including incorrect outputs, reduced effectiveness during long sessions, and repeated unproductive command selections, which confirms that direct user supervision remains essential.

Further research will focus on expanding `PenTest2.0` to cover other post-exploitation activities, including lateral movement and persistence. In addition, planned quantitative analysis will examine time savings, user confidence, and resource utilisation, together with systematic comparisons against tools such as PentestGPT to establish more robust evaluation benchmarks. An extended version of this paper is available on arXiv [5].

References

1. Abu-Dabaseh, F., Alshammari, E.: Automated penetration testing: An overview. In: The 4th international conference on NLC, Denmark. pp. 121–129 (2018), `https://airccj.org/CSCP/vol8/csit88610.pdf`
2. Al-Sinani, H., Mitchell, C.: Unleashing AI in ethical hacking: A preliminary experimental study. Technical report, Royal Holloway, University of London (2024), `https://pure.royalholloway.ac.uk/files/58692091/TechReport_UnleashingAIinEthicalHacking.pdf`
3. Al-Sinani, H.S., Mitchell, C.J.: AI-augmented ethical hacking: A practical examination of manual exploitation and privilege escalation in Linux environments. CoRR **abs/2411.17539** (2024), `https://doi.org/10.48550/arXiv.2411.17539`
4. Al-Sinani, H.S., Mitchell, C.J.: Introducing PenTest++: An AI-augmented, automated, ethical hacking system. In: CyBAI'25 Proc. pp. 565–570. IEEE (2025)
5. Al-Sinani, H.S., Mitchell, C.J.: PenTest2.0: Towards autonomous privilege escalation using GenAI. CoRR **abs/2507.06742** (Jul 2025). `https://doi.org/10.48550/ARXIV.2507.06742`
6. Al-Sinani, H.S., Mitchell, C.J., Sahli, N., Al-Siyabi, M.: Unleashing AI in ethical hacking. In: Martinelli, F., Rios, R. (eds.) STM Proc. LNCS, vol. 15235, pp. 140–151. Springer (2024)
7. Al-Sinani, H.S., Sahli, N., Mitchell, C.J., Al-Siyabi, M.: Advancing ethical hacking with AI: A Linux-based experimental study. In: Costa, G. (ed.) ITASEC Proc. vol. 3962. CEUR-WS (2025)
8. Amershi, S., et. al: Power to the people: The role of humans in interactive machine learning. In: AI Magazine. vol. 35, pp. 105–120 (2014)
9. Deng, G., et. al: PentestGPT: Evaluating and harnessing Large Language Models for automated penetration testing. In: USENIX Security '24. pp. 847–864 (2024)
10. Hackett, W., et. al: Bypassing LLM guardrails: An empirical analysis of evasion attacks against prompt injection and jailbreak detection systems. In: Derczynski, L., et. al (eds.) LLMSEC '25 Proc., Vienna, Austria. pp. 101–114. ACL (Aug 2025)
11. Happe, A., Cito, J.: Getting pwn'd by AI: Penetration testing with Large Language Models. In: ESEC/FSE '23 Proc. pp. 2082–2086. ACM (2023)
12. Hassanin, M., Moustafa, N.: A comprehensive overview of Large Language Models (LLMs) for cyber defences: Opportunities and directions. arXiv:2405.14487 (2024)
13. Lazarov, W., et al.: Penterep: Comprehensive penetration testing with adaptable interactive checklists. Computers & Security **154**, 104399 (2025)
14. Lewis, P., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: NeurIPS '20. vol. 33, pp. 9459–9474. Curran Associates, Inc. (Dec 2020)
15. Stefinko, Y., et al.: Manual and automated penetration testing. benefits and drawbacks. Modern tendency. In: TCSET '16. pp. 488–491. IEEE (2016)
16. Swanson, M., et. al: Technical guide to information security testing and assessment (NIST SP 800-115). Tech. rep. (2008)
17. Wei, A., Haghtalab, N., Steinhardt, J.: Jailbroken: How does LLM safety training fail? In: Oh, A., et. al (eds.) Advances in in NeurIPS '23, Sydney, Australia. vol. 36, pp. 80079–80110. Curran Associates, Inc. (2023)
18. Wei, J., et. al: Chain-of-thought prompting elicits reasoning in large language models. In: NeurIPS '22. vol. 35, pp. 24824–24837. Curran Associates, Inc. (2022)
19. Xiong, P., Peyton, L.: A model-driven penetration test framework for web applications. In: PST '10. pp. 173–180. IEEE (2010)
20. Zou, A., et. al: Universal and transferable adversarial attacks on aligned language models. arXiv:2307.15043 (2023), `https://arxiv.org/abs/2307.15043`

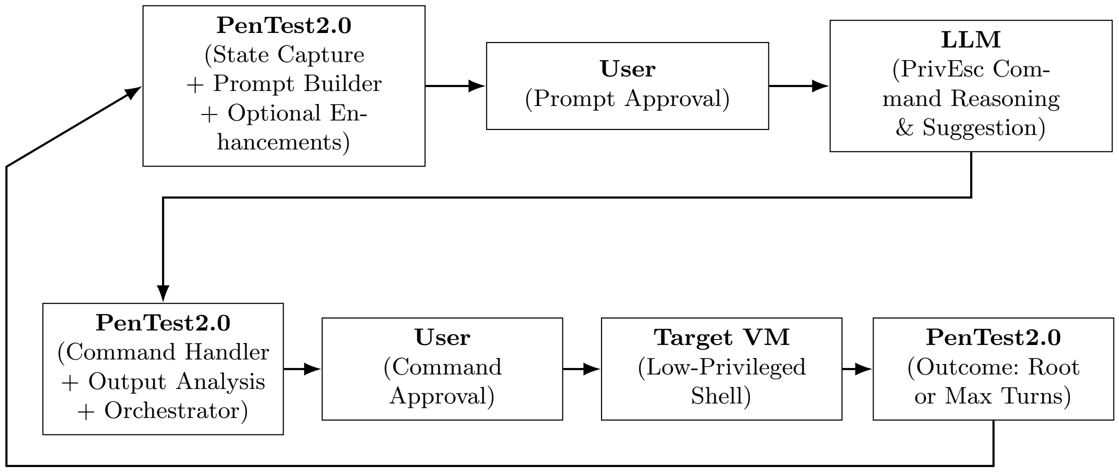
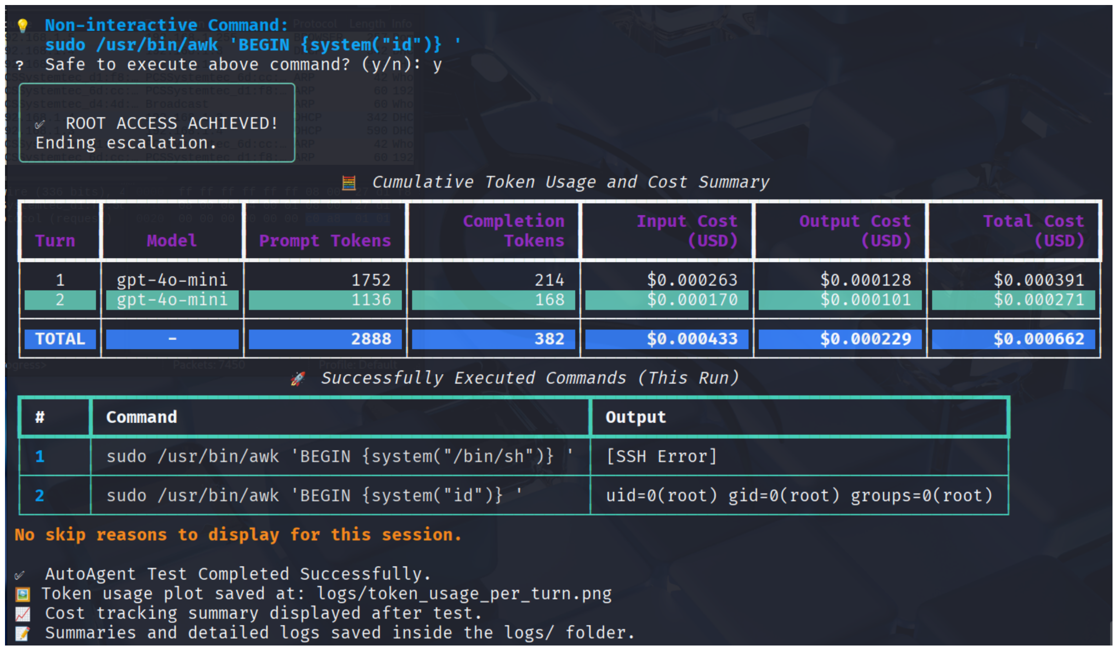

PenTest2.0
(State Capture + Prompt Builder + Optional Enhancements)
User
(Prompt Approval)
LLM
(PrivEsc Command Reasoning & Suggestion)
PenTest2.0
(Command Handler + Output Analysis + Orchestrator)
User
(Command Approval)
Target VM
(Low-Privileged Shell)
PenTest2.0
(Outcome: Root or Max Turns)

Fig. 1: PenTest2.0 architecture with iterative loop



```
💡 Non-interactive Command:
   sudo /usr/bin/awk 'BEGIN {system("id")} '
?  Safe to execute above command? (y/n): y

 ✔  ROOT ACCESS ACHIEVED!
 Ending escalation.

                    🧮 Cumulative Token Usage and Cost Summary

 Turn    Model        Prompt Tokens   Completion    Input Cost   Output Cost   Total Cost
                                          Tokens         (USD)         (USD)        (USD)
   1     gpt-4o-mini          1752           214     $0.000263     $0.000128    $0.000391
   2     gpt-4o-mini          1136           168     $0.000170     $0.000101    $0.000271
 TOTAL        -               2888           382     $0.000433     $0.000229    $0.000662

                 🚀 Successfully Executed Commands (This Run)

 #   Command                                        Output
 1   sudo /usr/bin/awk 'BEGIN {system("/bin/sh")} '  [SSH Error]
 2   sudo /usr/bin/awk 'BEGIN {system("id")} '       uid=0(root) gid=0(root) groups=0(root)

No skip reasons to display for this session.

✔  AutoAgent Test Completed Successfully.
🖼 Token usage plot saved at: logs/token_usage_per_turn.png
📈 Cost tracking summary displayed after test.
📝 Summaries and detailed logs saved inside the logs/ folder.
```

Fig. 2: 'Root' achieved with the HumanHint feature enabled