AI-Guided Network Infiltration: A Comparative Case Study of Pentestgpt-Guided and Manual Penetration Testing

Sathiya Varma Parmeswaran[1] and Mohd Nazrien Zaraini[2*]

[1]Fakulti Kecerdasan Buatan dan Keselamatan Siber, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

[2]Fakulti Teknologi Maklumat dan Komunikasi, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

***Corresponding Author**

DOI: https://doi.org/10.47772/IJRISS.2026.101000369

Received: 24 October 2026; Accepted: 29 October 2026; Published: 03 November 2026

ABSTRACT

Penetration testing is time consuming and requires expert knowledge. Most evaluations are based on benchmark targets and report aggregate scores, while large language models can help to speed it up. They rarely show the same tool working on one system and then failing on another. This paper presents a case study of comparing PentestGPT, an LLM-driven assistant using GPT-4o, with manual testing in four single-run simulations, set up in an isolated VirtualBox lab The same SSH and SUID privilege escalation chain was executed twice on Metasploitable 2: manually, and with the assistance of PentestGPT. Both runs were made using default credentials and a SUID nmap escape. The manual run took 21 minutes. The guided run took 15 minutes (29% less), with nearly all that savings coming from privilege escalation, as the tool was renamed the exploitable binary rather than searching for it. The same guided path was then applied to a modern Ubuntu Server. It did not transfer. There was no SUID binary of 'nmap' in Ubuntu, so the escape that was used on the legacy target would not run and the operator resorted to manual technique to verify root. A parallel SMTP enumeration was good in concept as it assumed less security on a server than Ubuntu had. The pattern is consistent across the four simulations. PentestGPT is fast and accurate where the attack surface is familiar and well documented. It is weaker where a path must be adapted to an unfamiliar system, and the human operator supplied that adaptation in every case. The study supports using LLMs as guided assistants under human oversight, not as autonomous testers.

Keywords: penetration testing, large language models, PentestGPT, human-in-the-loop, case study

INTRODUCTION

Penetration testing involves conducting simulated attacks on a system to evaluate its security. Established methodologies define structured phases from reconnaissance to reporting (Penetration Testing Execution Standard [PTES], n.d.; Scarfone et al., 2008). However, execution remains manual and hugely relies on the tester's expertise. That dependence limits scalability and consistency as infrastructures grow more complex.

Generative AI can help to ease that dependence. The idea of PentestGPT is to use a large language model with its parsing, reasoning and generation capabilities to instruct a human operator on a test (Deng et al., 2024). The results they report are impressive but are based on a standard or capture-the-flag scenario (Gioacchini et al., 2024). A benchmark score indicates that the tool functions on the benchmark. It doesn't tell you if a path that is working on one machine, will work on the next.

This paper deals with that gap. We run PentestGPT and a manual tester through the same attack on one legacy target, measure the difference, and then take the tool's own winning path onto a second, modern target to see whether it holds. The contributions are as follows:

1. **A matched manual-versus-guided comparison.** The same SSH and SUID attack chain on Metasploitable 2 is executed here manually and with the guidance of PentestGPT with the same conditions.

2. **A cross-target generalisation test.** PentestGPT is applied to a modern Ubuntu Server to test whether a working escalation path from one target carries over to another.

3. **A documented failure mode.** The study records where PentestGPT's suggestions needed manual substitution, and why.

The rest of the paper is organised as follows. The literature review includes penetration testing methodologies and LLM-assisted tools. The methodology describes the lab and the four simulations. The results report the timing and outcomes. The discussion interprets the pattern, and the conclusion closes the paper.

LITERATURE REVIEW

Penetration testing methodologies

Penetration testing follows published methodologies. NIST SP 800-115 describes technical security testing and assessment (Scarfone et al., 2008). PTES defines phases from pre-engagement to reporting, including exploitation and post-exploitation (PTES, n.d.). Both rely on the tester's judgement at each phase. That is why manual testing gives high contextual accuracy, but it is slow and depends heavily on the expert.

Automated and AI-assisted penetration testing

Automation came first through reinforcement learning. Zennaro and Erdődi (2020) modelled penetration testing as a reinforcement learning problem on capture-the-flag challenges and examined the trade-off between model-free learning and prior knowledge. Such agents need a formal model of the environment, and that limits real-world use.

LLM-based agents are the newer route. VulnBot uses several collaborating agents to run penetration tests autonomously (Kong et al., 2025). PentestAgent and AutoPentest also place LLM agents inside the testing workflow (Henke, 2025; Shen et al., 2025). AutoPenBench provides a benchmark for generative agents (Gioacchini et al., 2024). He et al. (2026) survey 81 papers from 2023 to 2026 on LLM-driven penetration testing agents. Most of this work targets autonomy and benchmark scores rather than a direct, matched comparison with manual testing.

LLM-assisted penetration testing tools

PentestGPT is the best-known tool in this group. It wraps an LLM with reasoning, generation and parsing modules and was evaluated on benchmark targets (Deng et al., 2024). Happe and Cito (2023) used LLMs as sparring partners in penetration testing. Happe et al. (2023) tested LLMs on Linux privilege escalation, the same phase examined in this study. PenHeal adds a second stage that recommends remediation after the test (Huang & Zhu, 2024). Ferrag et al. (2025) review LLM applications and vulnerabilities across cybersecurity more broadly.

Risks and human oversight

LLMs make mistakes with confidence. Huang et al. (2025) conduct a survey of hallucination in LLMs and categorize its types and causes. Prompt injection is a second risk. Greshake et al. (2023) demonstrated the vulnerability of real-world LLM-integrated applications against indirect prompt injection attacks. Both risks call for a human to review this output before use.

Human-in-the-loop (HITL) design puts that human inside the workflow. Chen et al. (2023) describe HITL machine learning and its applications in population health. Ostheimer et al. (2021) study hybrid intelligent systems where humans and machines complement each other. Nikitin and Kaski (2021) elicit decision rules from

people to support domain adaptation. These studies come from other fields. They motivate a HITL approach in offensive security, where the operator validates and executes every command.

Research gap

Initially the question was whether LLMs can run a penetration test on their own. Most recent work is done on benchmarks, and reports on how many times the agent gets to the goal. Two things are rarer. One is a matched comparison between manual and guided testing on the same target, so the time saving can be read directly. The other is a transfer test, taking a path that worked on one target onto a different one. This study does both, with a matched comparison on Metasploitable 2 and a transfer test onto Ubuntu Server.

METHODOLOGY

This study employs a case study approach that uses single-run simulations instead of multiple trials. Each simulation is based on the penetration testing lifecycle (recognition, enumeration, exploitation, post exploitation and reporting). As each scenario has been run only once, the analysis is run-level, rather than statistical.

Experimental environment

The experiment used Oracle VirtualBox with host-only networking (192.168.56.0/24) to isolate traffic. Kali Linux (192.168.56.10) was the attacker machine, running Nmap, Telnet, tcpdump and Wireshark. Metasploitable 2 (192.168.56.20) served as the deliberately vulnerable legacy target. Ubuntu Server (192.168.56.30) served as a modern, patched target with no intentionally planted vulnerability. Figure 1 shows the attacker environment used for both targets.

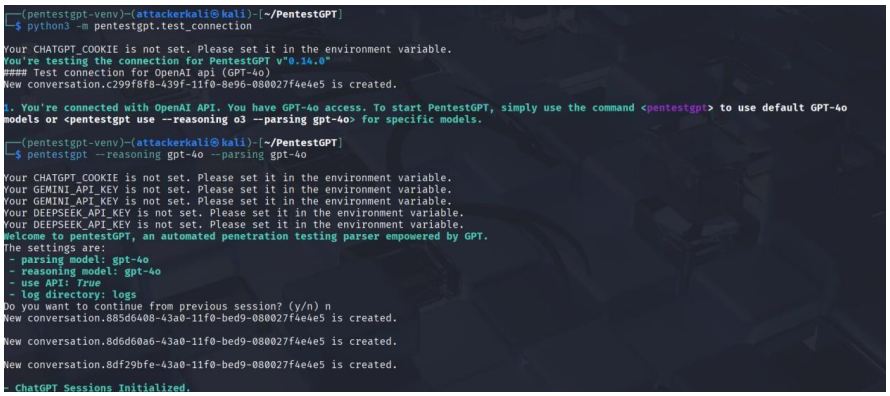


Figure 1: PentestGPT session on the Kali Linux attacker machine, confirming the OpenAI API connection and the active reasoning and parsing models.

LLM integration and toolchain

PentestGPT version 0.14.0 was used, configured with GPT-4o as both the reasoning and parsing model and connected through the OpenAI API. The tool maintains a running penetration testing tree and updates each task to completed when the operator marks it as such and logs each session with a timestamp.

The workflow was strictly human-in-the-loop. PentestGPT proposed the next command. It was checked for syntax and safety by the operator, and then it was run and the results were returned to PentestGPT, before the next suggestion.

Simulation design

Four single-run simulations were conducted:

- **Simulation 1 (PentestGPT-guided).** SSH and SUID privilege escalation on Metasploitable 2.

- **Simulation 2 (traditional).** The same SSH and SUID attack on Metasploitable 2, conducted manually as a benchmark.
- **Simulation 3 (PentestGPT-guided).** SMTP and Exim enumeration on Ubuntu Server.
- **Simulation 4 (PentestGPT-guided).** SSH and SUID privilege escalation on Ubuntu Server.

For each simulation, the outcome (success or failure), wall-clock time per stage from session transcript/timestamps, number of manual commands after reconnaissance, and label (directly executable, partially useful, or incorrect) for each AI recommendation were recorded. All of PentestGPT's recommendations were checked with the operator before implementation.

RESULTS

Metasploitable 2: manual versus PentestGPT-guided

Both runs reached root through the same chain. Reconnaissance with Nmap identified SSH on port 22. Login succeeded with the default credentials msfadmin:msfadmin, which gave user-level access. Privilege escalation used a SUID nmap binary already present on Metasploitable 2 to escape into a root shell, confirmed by inspecting /etc/shadow. In the manual run this binary was found by a general SUID search. In the PentestGPT-guided run the tool suggested the same binary and command directly.

Table 1 reports the stage-by-stage timing.

Table 1: Metasploitable 2 timing, manual versus PentestGPT-guided

Stage	Manual (min)	PentestGPT-guided (min)	Saving	Accuracy
Reconnaissance	5	5	None	Accurate, both
Enumeration	4	3	1 min	Accurate, both
Privilege escalation	10	5	5 min	Accurate, both (nmap --interactive)
Post-exploitation	2	2	None	Accurate, both
Total	**21**	**15**	**6 min (29%)**	

PentestGPT-guided testing completed the chain in 15 minutes against 21 minutes manually, a reduction of about 29%. The largest saving was in privilege escalation, where PentestGPT named the exploitable binary directly rather than requiring a general search. Figure 2 shows the binary listing from the manual SUID search.

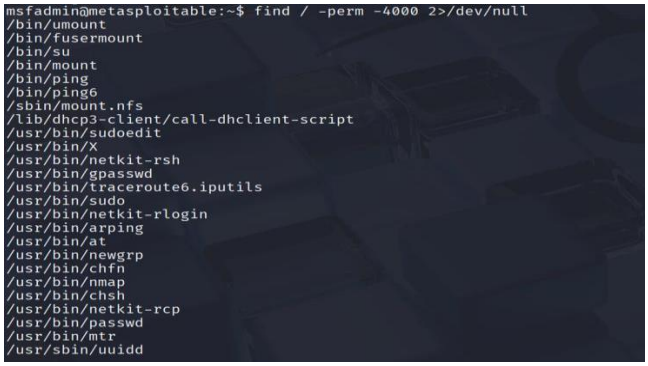


Figure 2: Binaries with the SUID bit set, found by the manual search "find / -perm -4000" on Metasploitable 2.

Ubuntu Server: SMTP enumeration

On Ubuntu Server, Nmap identified Exim 4.90 on port 25. PentestGPT suggested standard SMTP checks, including VRFY and EXPN commands and a relay test. Figure 3 shows the enumeration exchange.

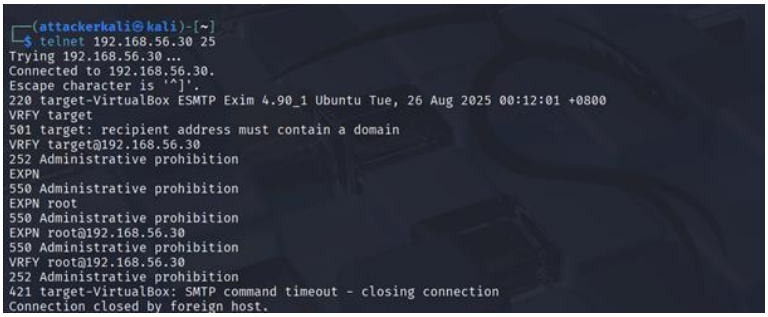


Figure 3: VRFY and EXPN enumeration against the Exim 4.90 service on Ubuntu Server, run manually over Telnet on PentestGPT's suggestion.

The VRFY and EXPN tests confirmed that local usernames could be verified, though several returned an administrative prohibition rather than a direct answer. A relay test from an external address to an external address was correctly denied. PentestGPT's suggestions assumed a looser default configuration than the server had, so some steps needed manual confirmation rather than being directly actionable. No privilege escalation was attempted in this simulation, since the objective was information disclosure and relay testing only.

Ubuntu Server: SSH and SUID attempt

Login again succeeded with msfadmin:msfadmin over SSH. A SUID search found passwd, su, sudo and related system binaries, but not nmap. The nmap --interactive escape that worked on Metasploitable 2 did not apply, because Ubuntu Server had no SUID nmap binary. The operator substituted a manual check of the standard SUID binaries and confirmed root access through them, which PentestGPT's Metasploitable 2-derived suggestion had not anticipated.

Table 2 summarises the outcome of all four simulations.

Table 2: Outcome summary across the four simulations

Simulation	Target	Outcome	PentestGPT role
1: SSH/SUID (guided)	Metasploitable 2	Root via default credentials and SUID nmap --interactive	Suggested the full chain; operator validated each step
2: SSH/SUID (manual)	Metasploitable 2	Root via the same chain, found by manual SUID search	Not used (benchmark run)
3: SMTP/Exim	Ubuntu Server	User enumeration confirmed; open relay denied	Suggested VRFY, EXPN and relay tests; some assumed defaults the server did not have
4: SSH/SUID	Ubuntu Server	Root via manual SUID binaries after the nmap path failed	Suggested the Metasploitable 2 chain; operator substituted passwd, su, sudo

DISCUSSION

Initially the Metasploitable 2 comparison looked like a clean win for PentestGPT. It cut total time by 29%, and nearly all of that came from one stage. In the manual run the tester searched for SUID binaries and read the list. PentestGPT skipped the search and named the binary. But that is the point to be careful about. The target was

deliberately vulnerable and its escalation path is documented everywhere. The tool was recalling a known answer, not working one out.

Ubuntu Server put that reading to the test. The same escalation suggestion came back, but Ubuntu had no SUID nmap binary to escape through, so the suggestion had nothing to act on. Recall had produced an answer that fit the wrong machine. The root shell on Ubuntu came from the operator's manual fallback, not from the tool. The SMTP simulation showed a milder version of the same thing. The enumeration plan was valid, but it was based on a loosen configuration than the server was running in, and the operator had to order steps that were applicable from the steps that were not.

So the value and the limit have the same source. PentestGPT is fast because it retrieves, and it is brittle for the same reason. It helps most where the attack surface is familiar and documented, and least where a path must be fitted to a system it was not drawn from. In all four simulations the operator supplied that fit. That is why the tool belongs in a human-in-the-loop role and not an autonomous one.

This study has several limitations. The first is that the evaluation was based on just four single-run simulations and no retrials. The observed 29% reduction in testing time therefore reflects a single comparison rather than a consistent performance improvement. Second, we have conducted experiments in just two target systems and mainly on SSH and SMTP services. The findings may not extend to other operating systems, network configurations or attack scenarios. Third, the simulations were all carried out by a single operator, and his experience and judgment may have affected the actual duration and the analysis of PentestGPT's recommendations. Lastly, the ability and configuration of the SUID binaries on each target affected the success of the privilege escalation methods. The findings must therefore be seen as an observation of the controlled case study, and not as a measure of PentestGPT's performance in general across various environments.

CONCLUSION

This case study compared PentestGPT-guided penetration testing with manual testing on the same legacy target, and then tested whether a working PentestGPT-guided path generalised to a modern target. On Metasploitable 2, PentestGPT reduced total time by 29% over a matched manual run, with the largest saving in privilege escalation. On Ubuntu Server, PentestGPT guided a sound SMTP enumeration and correctly identified SSH as a vector, but its Metasploitable 2-derived escalation suggestion did not apply, and the operator substituted a manual method.

The two results point the same way. A tool that retrieves known paths saves real time on known ground and stumbles the moment the ground changes. That is why PentestGPT belongs beside an operator who can tell the difference, not in place of one. Future work should add repeated runs and more varied targets to test how far the pattern holds.

Declarations

Ethical approval

All testing was carried out in an isolated virtual laboratory on systems built for this purpose. No real systems, human participants or personal data were involved.

AI usage disclosure

PentestGPT was used to suggest reconnaissance, exploitation and post-exploitation steps. All command execution, validation, environment configuration, data collection and result interpretation were performed by the human operator. No AI system autonomously executed attacks or generated experimental outcomes.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support provided by Universiti Teknikal Malaysia Melaka (UTeM)

REFERENCES

1. Chen, L., Wang, J., Guo, B., & Chen, L. (2023). Human-in-the-loop machine learning with applications for population health. CCF Transactions on Pervasive Computing and Interaction, 5(1), 1-12. https://doi.org/10.1007/s42486-022-00115-4
2. Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., & Rass, S. (2024). PentestGPT: Evaluating and harnessing large language models for automated penetration testing. Proceedings of the 33rd USENIX Security Symposium, 847-864. https://www.usenix.org/conference/usenixsecurity24/presentation/deng
3. Ferrag, M. A., Alwahedi, F., Battah, A., Cherif, B., Mechri, A., Tihanyi, N., Bisztray, T., & Debbah, M. (2025). Generative AI in cybersecurity: A comprehensive review of LLM applications and vulnerabilities. arXiv. https://arxiv.org/abs/2405.12750
4. Gioacchini, L., Mellia, M., Drago, I., Delsanto, A., Siracusano, G., & Bifulco, R. (2024). AutoPenBench: Benchmarking generative agents for penetration testing. arXiv. https://arxiv.org/abs/2410.03225
5. Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. arXiv. https://arxiv.org/abs/2302.12173
6. Happe, A., & Cito, J. (2023). Getting pwn'd by AI: Penetration testing with large language models. Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. https://doi.org/10.1145/3611643.3613083
7. Happe, A., Kaplan, A., & Cito, J. (2023). LLMs as hackers: Autonomous Linux privilege escalation attacks. arXiv. https://arxiv.org/abs/2310.11409
8. He, Z., Dong, J., Li, Z., Chen, T., Deng, G., Luo, F., Ji, J., Cao, Y., & Luo, X. (2026). A survey of LLM-driven penetration testing: Taxonomy, co-evolution, and open challenges. arXiv. https://arxiv.org/abs/2607.02605
9. Henke, J. (2025). AutoPentest: Enhancing vulnerability management with autonomous LLM agents. arXiv. https://arxiv.org/abs/2505.10321
10. Huang, J., & Zhu, Q. (2024). PenHeal: A two-stage LLM framework for automated pentesting and optimal remediation. arXiv. https://arxiv.org/abs/2407.17788
11. Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., & Liu, T. (2025). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Transactions on Information Systems. https://doi.org/10.1145/3703155
12. Kong, H., Hu, D., Ge, J., Li, L., Li, T., & Wu, B. (2025). VulnBot: Autonomous penetration testing for a multi-agent collaborative framework. arXiv. https://arxiv.org/abs/2501.13411
13. Nikitin, A., & Kaski, S. (2021). Decision rule elicitation for domain adaptation. Proceedings of the 26th International Conference on Intelligent User Interfaces. https://doi.org/10.1145/3397481.3450682
14. Ostheimer, J., Chowdhury, S., & Iqbal, S. (2021). An alliance of humans and machines for machine learning: Hybrid intelligent systems and their design principles. Technology in Society, 66, Article 101647.
15. Penetration Testing Execution Standard. (n.d.). The Penetration Testing Execution Standard. https://pentest-standard.readthedocs.io/
16. Scarfone, K., Souppaya, M., Cody, A., & Orebaugh, A. (2008). Technical guide to information security testing and assessment (NIST Special Publication 800-115). National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf
17. Shen, X., Wang, L., Li, Z., Chen, Y., Zhao, W., Sun, D., Wang, J., & Ruan, W. (2025). PentestAgent: Incorporating LLM agents to automated penetration testing. arXiv. https://arxiv.org/abs/2411.05185
18. Zennaro, F. M., & Erdődi, L. (2020). Modeling penetration testing with reinforcement learning using capture-the-flag challenges: Trade-offs between model-free learning and a priori knowledge. arXiv. https://arxiv.org/abs/2005.12632