# Let It Hack: Autonomous Multi-Agent Penetration Testing with LLMs and Tool-Augmented Reasoning

Stephen Lewis
Portsmouth University
Portsmouth, United Kingdom
stephen.lewis1@myport.ac.uk

Aikaterini Kanta
Portsmouth University
Portsmouth, United Kingdom
katerina.kanta@port.ac.uk

## ABSTRACT

Recent advances in large language models (LLMs) have opened new opportunities for automating complex tasks in Cybersecurity, including offensive operations. However, most existing approaches to LLM-assisted penetration testing rely on human input or scripted interactions. This work explores a fully autonomous, local-agent framework for end-to-end penetration testing, using LangGraph to structure multi-stage tool-augmented reasoning. Without human intervention post-launch, selected open-source LLMs were tasked with scanning, vulnerability analysis, and exploitation against a standard testbed. Results show that models such as Qwen-14B and Qwen-32B can successfully execute multiple real-world exploits, demonstrating that local, API-free LLM agents can move beyond advisory roles into operational offensive security.

## KEYWORDS

LLMs, Agents, Penetration Testing, Cybersecurity, Agentic AI, LangGraph, Large Language Models, GAAI

## 1 INTRODUCTION

Penetration testing, also known as ethical hacking, is a systematic process that involves attempting to breach a system's defences using various techniques such as network scanning, vulnerability exploitation, and social engineering. Penetration testing is a critical component of an organisation's cybersecurity strategy, enabling it to identify vulnerabilities, assess risks, and implement remediation measures before a malicious actor can exploit them. The primary objective of penetration testing is to simulate real-world attacks on computer systems, networks, or web applications to test their defences and identify potential entry points for attackers.

Penetration testing is essential for several reasons:

(1) *Vulnerability Identification*: Penetration testing helps identify vulnerabilities in systems, networks, and applications that could be exploited by attackers.
(2) *Risk assessment*: Penetration testing assesses the potential risks associated with identified vulnerabilities, allowing organisations to prioritise remediation efforts.
(3) *Compliance*: Regular penetration testing is often required for compliance with industry standards and regulations, such as PCI-DSS[1] and HIPAA[2].
(4) *Improved Security Posture*: Penetration testing helps organisations strengthen their security posture by identifying and addressing weaknesses and reducing the risk of successful attacks.

However, the increasing use of artificial intelligence by malicious actors is of particular concern. Afane et al. [1] describe how attackers are exploiting LLMs to create increasingly convincing phishing e-mails. They note, however, that LLMs have a dual use in this regard; they can be used by attackers to create malicious content, but they can also be successfully employed to detect phishing attempts.

A significant concern amongst penetration testers when using generative AI is the potential for unintentional disclosure of highly sensitive information through the use of online hosted services, such as ChatGPT. The use of locally hosted LLMs mitigates this risk by ensuring that data related to target systems remains on the local host. It is critical, therefore, to consider LLMs that do not send confidential data back to cloud-providers.

### 1.1 Contribution of this work

In this paper, a methodology for evaluating the use of locally run LLMs for penetration testing is presented. This work aims to show that locally run LLMs can act autonomously in identifying and exploiting vulnerabilities in systems. The methodology described in this paper presents an Agentic framework for evaluating LLMs by breaking down penetration testing into sub-tasks and asking LLM agents to search for vulnerabilities, then exploit them. This approach is then evaluated with different LLMs to compare the capabilities that they demonstrate.

To summarise, the contribution of this work includes:

- A technical evaluation of the reproducibility of previously published work relating to the use of generative AI in penetration testing,
- The development of a framework for testing locally run LLMs for penetration testing,
- A detailed evaluation of LLMs in completing penetration testing tasks.

The rest of the paper is organised as follows: Section 2 offers an overview of the related work in the field, Section 3 provides the methodology that was used to evaluate LLMs for penetration testing tasks. Section 4 shows the results of implementing this methodology, and finally, section 5 provides a summary and commentary of the results and outlines several avenues for future work.

[1]Payment Card Industry Data Security Standard
[2]Health Insurance Portability and Accountability Act

*Proc.@ of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/.* 2026.

## 2 RELATED WORK

Automated penetration testing using Large Language Models (LLMs) is a rapidly evolving field, with recent studies highlighting significant advances in the effectiveness and efficiency of LLMs in identifying vulnerabilities and simulating cyber attacks [6, 14]. Researchers have been actively exploring the potential of LLMs to automate various stages of the penetration testing process, including scanning, reconnaissance, exploitation, and post-exploitation activities.

One key finding from recent papers is the growing body of evidence supporting the use of LLMs as a viable tool for automated penetration testing. Authors such as Bianou and Batogna [5] have developed multi-agent frameworks using CrewAI[3], a rapid prototyping environment based on LangChain. Their goal is to develop specialised agents that can work on different stages of penetration testing, and use the MITRE ATT&CK [15] knowledge base to guide GPT4o LLM agents. Aslaner [4] notes that the ATT&CK framework is one of the most widely used and comprehensive Tactics, Techniques and Procedures (TTP) frameworks in the cybersecurity industry, and that, many security teams use it as their primary resource for TTPs.

Arguably, the most mature work on automated penetration testing is provided by Deng et al. [6]. Their software uses the GPT-4o LLM in chat mode to guide a penetration tester through the tasks required to carry out an end-to-end test. They adopt a novel approach in that the software creates a Pentesting Task Tree (PTT) at the start of the test (i.e. by using *nmap* to identify accessible services) and then uses the LLM to create sub-tasks [6]. The user then assists the LLM in working through the sub-tasks to identify vulnerabilities and, where possible, exploit them. They adopt a chat interface because LLMs are text-based and cannot independently perform penetration testing operations [6]. Other authors, however, are not in agreement with this and have been more successful with full automation [14]. Furthermore, one of the drawbacks to using commercial models is the cost. In their benchmark study, Deng et al. [6] write that PentestGPT completed 4 out of 10 penetration testing challenges, incurring a total cost of USD 131.5. Although they emphasise the potential of LLMs, they do also acknowledge their limitation. They concede that while LLMs can produce insightful outputs, their outcomes sometimes require revisions and explicitly provide for this in their design [6].

A study by Wu et al. [16] demonstrates that LLM-based Agents can simulate potential attackers and identify vulnerabilities that are exposed by interfaces and services. Similarly, Kong et al. [14] proposed a multi-agent collaborative framework for autonomous penetration testing, which leveraged LLMs to automate various stages of the testing process. These studies highlight the potential of LLMs to improve the efficiency and effectiveness of penetration testing, particularly in scenarios where manual testing is too time-consuming or impractical.

Gregory and Liao [8] adopt a different approach. Rather than use OpenAI commercial models, they use Mistral 7B running on a local machine. This uses retrieval augmented generation (RAG), a way of increasing the knowledge of LLMs from a database loaded with documents relevant to penetration testing, along with Low-Rank Adaptation (LoRA) for fine-tuning parameters within the model. They note that LoRA is an effective method for fine-tuning small models like Meta AI's Llama-7B for domain-specific tasks[8]. It is assumed, but not stated directly, that fine-tuning is necessary for circumventing the ethical controls that are built into the model.

Another significant finding from a recent research paper is the importance of prompt engineering in enhancing the performance of LLMs in penetration testing contexts.Antar [3] highlights the role of prompt-driven approaches in optimising the effectiveness and scalability of LLM-driven penetration testing. This is particularly evident when combined with frameworks like PR0MPTP1LOT (an intelligent, modular framework that applies LLMs to analyse console outputs, extract critical insights, and provide actionable guidance throughout the penetration testing lifecycle) and environments such as E.C.H.O. [3]. This highlights that careful design and engineering of prompts can significantly improve the accuracy and efficiency of LLMs in identifying vulnerabilities.

A study by Anisuzzaman et al. [2] further demonstrates that fine-tuning LLMs with specialised use cases can lead to significant improvements in performance, particularly in scenarios where the model is required to generate high-quality payloads or exploits. The authors achieved a level of success in customising existing LLMs using specialised data sets, selecting specific pre-trained LLMs, fine-tuning with hyperparameters, output validation, and using prompt examples. They note, however, a number of problems remain in using LLMs for specialised uses, including hallucinations, legal and safety concerns, biases that stem from training data sets, a lack of domain-specific data and data leakage.

Happe et al. [10] focus their attention on the use of LLMs for automated privilege escalation. Often, in a penetration test, access to a system is achieved first with a user identity that does not have full system access. Privilege escalation is a step in which the attacker gains root or admin access by exploiting a vulnerability within the system. A particularly interesting aspect of their research is their view that smaller LLMs are not suitable for penetration testing [10]. They argue that untuned small LLMs such as Llama3-8b are currently not feasible for penetration testing. In their work, they create a number of benchmark privilege escalation scenarios and compare different LLMs, along with human pen-testers. They found that humans could achieve a 75% success rate without hints or guidance, and 92% with guidance. GPT-3.5-turbo could only achieve a 16% success rate without guidance, but this increased to 50% with human guidance. GPT-4-turbo was much more capable, with a 66% success rate without guidance, and 83% with. Llama3-8b achieves 0% success without guidance, and 16% with guidance. Their explanation for the superior performance of GPT-4-Turbo is its larger context window size (128k tokens), which can provide the model with much more information to base its reasoning on. GPT-3.5-turbo and the Llama3 models are limited to 8k tokens.

Wu et al. [16] are somewhat more ambitious with an attempt to create fully autonomous, end-to-end penetration testing software. Their approach is to use a state machine, based on the LangGraph framework, to go through the individual stages of penetration testing: scanning, selection of vulnerabilities, reconnaissance of specific vulnerabilities, exploitation and a check to see if the exploit was successful. Although the authors state that their pre-experimental

[3]https://www.crewai.com/

code is available in GitHub[4], this currently appears to be quite limited in functionality.

Their results are evaluated by testing their software on twenty simple CVE vulnerabilities. They write that GPT-4o mini successfully exploits six out of the twenty (30%), whereas GPT-3.5 only manages one [16]. This is consistent with other studies and may be related to context size and/or the number of training parameters. Although the authors mention the Llama3 family of models, they did not include these in their benchmark tests.

An interesting approach is adopted by [14]. In attempting to overcome the weaknesses of LLMs, they adopt a collaborative multi-agent approach. Essentially, they use three separate LLM Agents: a Planner, a Generator and a Summarizer. By using the Summariser Agent-LLM they attempt to make sense of that data, feeding back into the Planner and the Generator Agents. In a similar way to how [6] describe their work above, they create a task tree which they refer to as a Penetration Task Graph (PTG) [14], which tracks the tasks to be carried out. As the graph progresses and new information about the target system is discovered, it is added to the tree. The authors claim considerable success in using local models, specifically Llama3.3-70B and Llama3.1-405B. They also note that automated penetration testing generates a large amount of unstructured data but often fails to provide actionable insights or provide next steps [14]. The authors also benchmark their software against AutoPenBench [7], and PentestGPT [6]. With Llama3.1-405B they claim an impressive 69.05% success rate, compared to 40% for PentestGPT with the same LLM. Even with the much smaller Llama3.3-70B they state that they can solve 59.52% of penetration tasks (34.76% for PentestGPT) [14].

Despite clear progress with LLMs in these research projects, recent papers also identify several challenges and limitations associated with the use of LLMs for automated penetration testing. For example, the study by [14] noted that open-source LLMs often struggle with tasks requiring high-level reasoning and creativity, such as privilege escalation and exploitation of complex vulnerabilities. Moreover, analysis by [12] revealed that even state-of-the-art LLMs such as GPT-4o and Llama3.1-405B encounter difficulties when faced with machines or tasks that require a nuanced understanding of contextual information.

To address these challenges, other studies have proposed several approaches to improve the performance of LLMs in penetration testing contexts. Bianou and Batogna [5] proposed using multi-agent frameworks for penetration testing automation. Multi-agent architectures allow multiple LLM Agents that have specialised abilities to collaborate on a task. By focusing on a specific sub-task, many of the weaknesses that LLMs experience can be overcome. The authors write that they have developed a proof of concept based on the GPT4.o model that can accurately identify vulnerabilities, suggest exploits, and report on these. In their view, however, part of the success can be attributed to GPT4.o having a high level of 'built-in' cybersecurity knowledge [5].

The precision and effectiveness of LLM-based penetration testing tools appear to vary significantly depending on the size of the LLM (number of parameters), the quality of the original LLM training data, the type of architecture or framework used and the specific use case. Problems more general to LLMs that compromise the ability of the algorithm to complete the task are described in the following sections.

[4]https://github.com/Dizzy-K/AutoPT

## 2.1 Loss of Context

LLMs often struggle to maintain long-term memory, which is crucial for linking vulnerabilities and developing effective exploitation strategies [6]. This loss of context leads to the model losing awareness of previous test outcomes, resulting in missed past results. Furthermore, research has shown that this issue can be exacerbated by the complexity of the testing environment, with LLMs struggling to keep track of multiple vulnerabilities and exploits. Furthermore, LLMs have difficulty in maintaining knowledge of the overarching testing scenario [6]. The authors also mention that in their benchmark, loss of *session context* was the most common reason for failures [6]. This can occur when LLMs prioritise recent information above older information, or when older information drops out of the context window.

## 2.2 Hallucinations

LLMs are statistical algorithms that produce output that the highest probability, based on their training. This means that they can produce results that are plausible statistically, but are factually incorrect. These results, when produced by LLMs, are known as hallucinations. Wu et al. [16] mention that LLM-based agents often output incorrect commands that cause task failures, which is an important aspect of optimising end-to-end penetration testing goals. Furthermore, Deng et al. [6] note that LLMs may generate inaccurate operations or commands, often stemming from inherent inaccuracies and hallucinations. As mentioned above, however, Kong et al. [14] claim that they are able to significantly reduce hallucinations by adding a vector database for RAG.

LLMs are trained using data, mostly from public domain sources. This can also lead to 'knowledge' that differs from professional experts. Deng et al. [6] note that for services that require password authentication, LLMs typically advise brute-forcing access, which is considered as a widely ineffective strategy [13]. Deng et al. [6] go on to highlight that LLMs likely learn these techniques from public domain reports about enterprise breaches. They also note that besides brute force, LLMs can also suggest that testers engage in CVE studies, SQL injections, and command injections. These recommendations are common, as real-world penetration testers often discuss these techniques, even though they may not always provide the right solution.

## 2.3 Ethical Protection

In their extensive review of the current state of the use of LLMs in Cybersecurity, Hasanov et al. [11] note that LLMs have built-in mechanisms designed to prevent access to illegal or offensive data (that stems from the training process). From the point of view of the LLM, penetration testing as an activity is not very different from hacking, therefore providing instructions to a penetration tester is something that should be censored. This can be demonstrated simply by asking an LLM to assist with gaining access to a computer system. It is however possible to bypass these safeguards with *jailbreaking* mechanisms. Hasanov et al. [11] mention prompt

injection as one such mechanism (carefully crafting prompts in such a way as to mislead the LLM about the actual activity being carried out). Modified models, known as de-censored or uncensored models are widely available from websites such as Huggingface[5]. However, Happe et al. [10] argue that locally run, ethical-free LLMs are not yet sophisticated enough to support penetration testing, unlike their commercial counterparts, such as GPT-4-Turbo. On the other hand, GPT-4-Turbo also includes ethics filters, which attempt to restrict its use for ethical purposes.

## 3 METHODOLOGY

The purpose of the study is to evaluate the extent to which LLMs can support penetration testers. As highlighted by the literature review, the knowledge gap is clear, and it is possible to formalise this into two research questions:

**RQ1:** Can LLMs autonomously carry out penetration tests on host computers?

**RQ2:** Are there qualitative differences in the abilities of different LLMs to carry out penetration tests?

In order to answer these, the approach adopted is to develop an LLM agnostic penetration testing tool, based on the insights gathered from the literature review and using up-to-date tools and models. This will be used to test a number of LLMs to determine how well they perform when used by the tool.

Following extensive research into the frameworks available, it was decided to implement the software using LangGraph. This was the approach followed by [16] in their study. LangGraph provides a framework for constructing multi-agent LLM systems, in which each node in the graph can instantiate an LLM with capabilities unique to that node. For instance, some nodes may have access to tools. It is also possible to use different LLMs at each node if that is advantageous. The path through the graph, defined as edges, is under either programmer or LLM control, which aligns well with the staged approach to penetration testing that is commonly seen. Although LangGraph supports commercial LLMs, such as ChatGPT, this study will focus on local LLMs running within Ollama, a software framework which can provide common APIs to LangGraph.

### 3.1 LLM Selection

LLMs were selected for the experiment if they were released in the last six months, supported tool calling, and contained fewer than 70B. The selected LLMs were of different types and parameter sizes and the complete list is shown on Table 1. The only difference will be a single parameter that selects the model. Once started, the tool will run without human intervention. However, since the behaviour of LLMs is essentially non-deterministic, it is necessary to give each LLM multiple opportunities to complete their tasks. The exploitation stage therefore is run up to three times (if the exploit has not been successful). LLM prompting will be substantially neutral. LLMs will be given guidance on which stage of the penetration testing they are contributing towards, but will not be told which tools to use, or which vulnerabilities to attempt to exploit.

[5]https://huggingface.co/

**Table 1: LLMs Selected For Testing**

| LLM | Parameters | Context Size |
|---|---|---|
| granite3.3:2b | 2B | 128k |
| granite3.3:8b | 8B | 128k |
| qwen3:4b | 4B | 40k |
| magistral:24b | 24B | 39K |
| llama3.1:8b | 8B | 128k |
| llama3-groq-tool-use:8b | 8B | 8K |
| llama3.2:3b | 3B | 128k |
| llama3.3:70b | 70B | 128k |
| llama4:16x17b | 17B | 10M |
| mistral-small3.2 | 24B | 128k |
| mixtral:8x22b | 22B | 64K |
| qwen3:8b | 8B | 40k |
| qwen3:14b | 14B | 40k |
| qwen3:32b | 32B | 40k |
| qwq | 32B | 40k |
| phi4-mini:3.8b | 3.8B | 128k |

### 3.2 Penetration Testing Process

Professional penetration testers typically follow a staged methodology. While different methodologies are applicable to different scenarios, the Cyber Kill Chain[6] as developed by Lockheed Martin, an American defence and aerospace company, is one of the most popular. It proposes seven stages for penetration testing:

(1) **Reconnaissance:** Collecting information about the target, such as email addresses, detecting open ports, and assessing accessible services,
(2) **Weaponisation:** Analysing information from the previous stage, identifying known vulnerabilities and constructing exploits,
(3) **Delivery:** Sending the exploit or payload to the target system,
(4) **Exploitation:** Exploiting the vulnerability in order to execute code on the target system,
(5) **Installation:** Installing malware on the target system;
(6) **Command and Control:** Creating a permanent communication channel for remote monitoring or control,
(7) **Action on Objectives:** Completing the original objective.

For the present evaluation, Metasploitable 2 is used as the penetration target host. It was released by Rapid7, the developers of Metasploit, as a training VM for students developing their penetration testing skills. Although Metasplotable 2 does not represent difficult penetration test scenarios, it is certainly the case that real-world attacks often use basic, overlooked, vulnerabilities. It is also the case that if simple penetration tests can be automated, then experienced (human) penetration testing resources can be better reserved for testing more complex scenarios.

We argue that the insights gained from this experiment will be extremely valuable in the subsequent development of the software.

[6]https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

A guide on the Rapid7 website[7] explains the main vulnerabilities and how to exploit them. Table 2 provides an overview of these. The documented vulnerabilities on Metasploitable 2 range from easy to medium difficulty, and can typically be exploited in one discrete step. An unknown number of un-documented vulnerabilities also exit. The test LLMs successfully exploited some of these.

**Table 2: Exploitable vulnerabilities in Metasploitable 2**

| Ref | Port | Description |
|---|---|---|
| Vuln1 | 21 | VSFTP server version 2.3.4 has a backdoor that can open port 6200 with a root shell |
| Vuln2 | 22 | There are a number of user accounts with weak passwords that can allow access over SSH |
| Vuln3 | 80 | Numerous web server vulnerabilities |
| Vuln4 | 139/445 | Samba can be exploited to provide root access to the file system |
| Vuln5 | 513 | rlogin can be exploited to gain root access without a password |
| Vuln6 | 1524 | Has an "ingreslock" backdoor |
| Vuln7 | 2049 | Can use NFS to mount the root file system |
| Vuln8 | 3306 | mysql allows root access without a password |
| Vuln9 | 3632 | distccd backdoor |
| Vuln10 | 5432 | Weak password for Postgres |
| Vuln11 | 5900 | VNC allows access without password |
| Vuln12 | 6667 | UnrealRCD IRC daemon has a backdoor |

Without being able to understand graphical displays, it is unlikely that LLMs will be able to make sense of what would appear in a web browser (Vuln3) or the VNC service (Vuln11), so excluding these, there are ten documented exploitable vulnerabilities in the test. The LLMs under test are expected to find between zero and ten of these.

The software design uses the Ollama APIs to run local LLMs. Once Ollama has been installed on the test machine, it is straightforward to download an LLM by simply running the command *ollama pull <model_name>*. Ollama then makes the model available to other applications using APIs. LangGraph has Ollama libraries that make use of these APIs. The Ollama website[8], provides an overview of the size and capabilities of each model. It is also possible to convert other models, such as Huggingface models, to the GGUF format (Georgi Gerganov Unified Format) used by Ollama.

## 3.3 Automated Penetration System Design

Figure 1 show a flow diagram for the target implementation. Dedicated LLMs are instantiated at each stage in order to complete the allocated task associated with that stage. The graph repeats RAG Search, Vulnerability Scan and Port Level Exploitation for every open port that is discovered in the target host.

At key stages in the process, information is stored in the database (DB). This is persistent so that the graph can be stopped and resumed if required. Since it may be the case that some tests require more than one vulnerability to be exploited simultaneously, the final stage is a System Level Exploitation, which takes the summary reports from the Port Level Exploitation stages and asks the LLM agent at this stage to attempt to use multiple exploits to gain access. The main components of the design are described below.

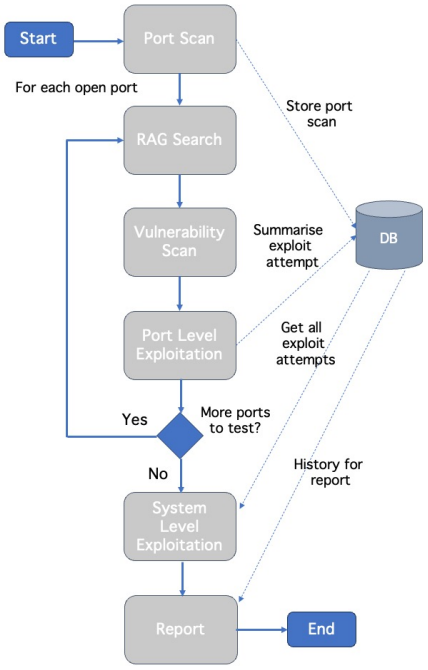


**Figure 1: High level design of the system**

## 3.4 Memory

LLMs do not have built in memory of prior interactions. However, LangGraph provides State, which can include a list of prior messages (inputs and outputs of the LLM call). State is passed between nodes, and can be manipulated as required. To allow the Agent to focus on the task at hand, it is useful to remove messages that are not relevant to the current task. So, for example, once the Port Scan node is complete, State will include details of all open ports. This can confuse the LLM if it is asked to focus on a specific port. To solve this problem, the list of ports and services is stored in a persistent datastore ('DB') and State messages are deleted when moving to the RAG Search node. As the graph proceeds, relevant messages are stored to the DB store, and retrieved only when they are needed.

## 3.5 Port Scan

The first Agent is provided with the *ip* address of the target host and instructed to carry out a port scan. It can decide to use any *kalilinux* tools via a Tools call. If the Agent completes its vulnerability scan, a second LLM call will structure the output into a JSON list of open ports, with the port number and a description of the services running behind that port. This is stored in the database so that the

[7] https://docs.rapid7.com/metasploit/metasploitable-2-exploitability-guide/
[8] https://ollama.com/search

program can be restarted at any point. The list will be used to loop the following nodes, one port at a time.

### 3.6 RAG Search

A RAG vector store is loaded with relevant information on penetration testing. This includes TTP that might help the Agent to successfully manage the exploitation task assigned. Sources such as webpages and PDF documents are loaded into the vector store as vector embeddings. A description of the port, for example, *Port 21 ftp vsftpd 2.3.4*, is provided as the input to the RAG query. The Agent then carries out a vector search and provides potentially relevant document fragments, which are passed to as context to the Vulnerability Scan Agent.

### 3.7 Vulnerability Scan

The Vulnerability Scan Agent has the ability to run Kali Linux commands to probe the services running on a specific port for vulnerabilities. It has full control of when it has gathered sufficient data, and moves on to the next node. The results of the vulnerability scan are stored in Langgraph State and are available to the Port Level Exploitation node.

### 3.8 Port Level Exploitation

The Port Level Exploitation Agent has the ability to run Kali Linux commands, and uses Tavily[9] to connect and search for information on the internet. It has the task of using information available from the following sources to reason about how it can get root access on the target host, given the tools available to it, and then gain access. The information sources are:

(1) Its own training data,
(2) the RAG search from the previous step,
(3) the vulnerability scan from the previous step,
(4) information returned from an internet search.

The purpose of this Agent is to find vulnerabilities, and attempt to exploit them to gain root access to the host, via the services running on the specified port. In an attempt to limit the effects of built-in LLM censorship control, the exploit prompt will be crafted to indicate that the activity is a CTF (Capture the Flag) competition.

### 3.9 System Level Exploitation

It is acknowledged that some exploits require simultaneous exploitation of vulnerabilities, on multiple ports. Once the software tool has completed scanning and exploitation attempts on each individual report, a final stage is for an LLM with summary knowledge of what has been tried so far, to attempt exploitation using multiple services and steps. This capability has not been tested in this experiment.

### 3.10 Report

This node retrieves data from the database and provides a summary report to the user. It includes any ports where an exploit was successful. The proposed software design attempts to mirror these stages with a graph node for each stage.

[9]https://www.tavily.com/

## 4 RESULTS

To evaluate the capabilities of each LLM, a LangGraph framework and penetration test target were configured. Minor differences in prompting were used to avoid LLM specific problems. In the results, a wide variation in the number of vulnerabilities were identified and successfully exploited by the LLMs being tested was observed. The definition used for a successful exploit is obtaining root permissions on the penetration test target via the identified vulnerability. The results can be found in Table 3.

The most successful models, qwen3:14b and qwen3:32b, were able to exploit four vulnerabilities out of the original list. Additionally, qwen3:32b also discovered vulnerabilities that were not documented (and therefore may have been included as training data). This is comparable with the work of Deng et al. [6] who claim success with four out of ten tasks. Their PentestGPT software depended, however, on a human to interpret the output and manually run external tools.

**Table 3: Exploits achieved by each evaluated LLM**

| LLM | V1 | V4 | V5 | V6 | V7 | Addl | Dur |
|---|---|---|---|---|---|---|---|
| granite3.3:2b | Y | | | | | | 00:20 |
| mistral-small3.2 | Y | Y | | | | | 03:26 |
| qwen3:4b | | | | | | | 04:03 |
| qwen3:8b | | | | Y | | | 04:32 |
| qwen3:14b | Y | Y | Y | | Y[1] | | 10:09 |
| qwen3:32b | Y | Y[2] | | Y | | Y[3] | 46:57 |
| qwq | | | | | | | 27:10 |

'Dur' the duration (elapsed time) of the test, in HH:MM.
Y[1] Whilst attempting to exploit potential vulnerabilities on port 52213, qwen3:14b successfully mounted the Metasploit 2 volume with root access (a port 1524 vulnerability).
Y[2] Accessed the Metasploit 2 volume using SMB as root and uploaded a reverse shell script, but failed to run it.
Y[3] Discovered an undocumented exploit of Java RMI via port 1099.

Several LLMs were not able to produce results. These are described in Table 4.

**Table 4: Unsuccessful LLMs**

| LLM | Description |
|---|---|
| granite3.3:8b | Does not call tools and instead tries to make up results. |
| llama3.2:3b | Gets stuck at the first tool call node by repeatedly iterating |
| llama3.3:70b | Model would not run, possibly not enough memory |
| llama4:16x17b | Was not able to make tool calls and would only provide instructions for manually running commands |
| phi4-mini:3.8b | Does not call tools and instead tries to make up results. |

As expected, none of the LLMs were able to handle graphical content and failed to make any progress with the vulnerabilities on

port 80 web services, or the VNC server. Overall the SSH Command Tool performed well where a shell was opened on the target host. However, it failed to detect where the remote disk was mounted on the Kali Linux host. Detection of the successful exploits in $Y^1$ and $Y^3$ was carried out through later review of the log files. Qwen3:14b was able to successfully mount the file system as root, and demonstrates that it has root access by doing cat on /etc/shadow (a file which contains password hashes, and is only possible for a root user).

LLMs were instructed to interact with the host machine using tool calls to a Kali Linux VM root command shell. This allowed the LLMs to interact with the target host in a controlled way, but also gave them significant scope for determining their approach. A number of optimisations were also included in the tool interface to improve SSH interaction:

(1) The session handler has a list of common Linux commands and sets timeouts accordingly. This means that, for example, *nmap* will wait longer for the command to complete, than ftp will. It also means that an attempt to run Hydra with the *rockyou.txt* password list will not result in the evaluation stopping for several days.
(2) Where a Metasploit exploit is attempted, there is hard-coded logic to check if a session is opened, and if so to send *whoami* and look for root to determine if the exploit worked.
(3) If a *nc*, *ssh* or *telnet* command are run, the tool looks to see if a *root@* prompt is found from the target host. If so, it sends *whoami* and looks for root to determine if a root shell has been established to the target host.
(4) A very large command response will be truncated. In testing it was observed that a *searchsploit ftp* request generated a response of over 90k characters. This caused problems for subsequent LLM calls. Responses over 10k characters therefore are truncated to the last 10k characters.
(5) Non-printable control characters, such as colour instructions in the shell, are removed from the response.

If root access to the host is achieved, the tool will add a message, "YOU HAVE ROOT ACCESS TO THE HOST" to the message state to alert the LLM Agent.

# 5 DISCUSSION

The study has shown that two LLMs in particular, when used within a multi-agent graph framework, are highly capable of autonomous penetration testing. The Qwen3:14b and Qwen3:32b models were able to carry out a scan of the target host, correctly assess open ports for vulnerabilities and work out how to exploit them. They both used a variety of techniques to do this, starting with a RAG query of preloaded penetration test data sources, supplementing this with additional scans of the host, carrying out active research of internet sources and using the custom SSH Command Tool to exploit the vulnerability. They used the *msfconsole*, *searchsploit*, *nmap* and *netcat* tools, as well as standard Linux utilities such as *mount*, *cat* and *rlogin* to achieve their objective.

## 5.1 Comparison to Existing Approaches

These LLMs, when used within the provided framework, performed better than the software that has been written about by other researchers, within the test parameters of the current study. A comparison to state-of-the-art approaches is presented:

- AutoPT [16] appears to be written to use a third party penetration tool, so it is questionable whether it was designed to demonstrate the ability of the LLM to perform the test. However, the results achieved in our experiment are better than the 30% success rate claimed on GPT-4o,
- PentestGPT [6] achieved good results (40% success rate) in the authors' tests but a human assistant was mandatory, which is not the case in this paper. Our attempt to replicate the results of this study was however unsuccessful,
- Vulnbot [14] was able to find one vulnerability in its scan, but failed to exploit it due to weaknesses in the tool design,
- Wintermute [10] had a much narrower scope, focusing on privilege escalation (assuming that access had already been achieved). In this case as well we were not able to replicate this result on the test host.

Moreover, unlike [9], this study shows that local ethically-restricted LLMs are indeed sophisticated enough to support penetration tasks, and appear to compete well with their commercial counterparts. Unfortunately, it was not possible to provide a comparison against the 69.05% success rate claimed by [14] with Llama3.1-405B because of the size of the LLM, and the limited computing resources available in this study.

## 5.2 Performance of the different LLMs

Only seven out of the fifteen LLMs tested completed the test, and these were the ones that did not face issues such as difficulty making tool calls, memory issues or intentional blocks resulting from ethical protection. It is possible that the other LLMs would have been able to carry out penetration test tasks if these protections were disabled, but this was not part of the scope of the present study.

The LLMs that successfully completed the test, successfully exploited between zero and four exploits. There is strong correlation between the capability to correctly identify and exploit vulnerabilities, and the number of parameters. However, qwq and Mistral-small3.2 did not do well, despite their size (32B and 24B), suggesting that LLM design and training is of relevance.

Aligning with the assumption that the Qwen3 models have similar internal structures and training, but differentiating on parameter size, it can be seen that the 32B and 14B models perform best, followed by the 8B model, then the 4B model. However, as the number of parameters increases, so does the computing cost. The 4B model took 4:03 hours, 8B was 04:32 hours (both running exclusively on the GPU), 14B was 10:09 and 32B was 46 hours and 57 minutes. In terms of efficiency, therefore the 14B model clearly outperformed all of the others. There is strong evidence, therefore, that there is a qualitative difference between LLMs in their applicability for penetration testing.

## 5.3 Accepted Limitations

The primary limitation of the current study is the use of a training VM, Metasploitable 2, as the target host, as this is not representative

of complex, real-world scenarios. However, it should be considered that many successful attacks exploit extremely simple vulnerabilities that would have been discovered by even a very junior penetration tester, had a test been carried out.

A more significant consideration in the use of Metasploitable 2 is that there are numerous guides on the internet that describe in detail each of the vulnerabilities specific to this machine. Therefore, there is a high liklihood that these guides will have formed part of the LLM's training set. Unfortunately, this is true for other publicly available intentionally vulnerable machines. HackTheBox and VulnHub training machines also have easily findable walk-throughs on the internet.

That some LLMs were able to exploit *undocumented* vulnerabilities on Metasploitable 2 is evidence of genuine utility, however.

Metasploitable 2 is a good choice for this study because it has multiple exploits, so reduces the complexity of setting up different vulnerable machines for different tests.

The study is also be limited to the use of text-based Kali Linux tools (command-line tools). This means that software such as Burp Suite, a tool for carrying out penetration testing on web servers, will not be available to the Agents. We plan to address this in future work, using multi-modal LLMs.

Finally, the capabilities of the test machine that was available was limited. This was an Intel Core i9 (10th Gen) machine with 96GB of DDR4 RAM and an Nvidia RTX 4060 Ti 16GB GPU. This limited the use of models with more than about 50 billion parameters.

## 5.4 Future Work

The software developed to test the LLMs, as described in this report shows considerable potential. There is much more, though, that can be added to improve it.

(1) The tests show that improvements in the SSH Command Tool can be made. Although it was good at identifying root access to the target when a shell was opened, it did not alert the Agent that the file system of the target was mounted with root access. This is a relatively small upgrade that could be made,
(2) Those models that do support tools, but do not appear to be very good at providing runnable Linux commands, often provide descriptions of what should be done. For instance, qwq shows that it describes how to execute a Metasploit exploit, but does not attempt to do so itself. It may be possible to use a small LLM (such as Qwen3:4b) to interpret the output of Mistral for the tool call. This would be possible because LangGraph allows each Agent to be based on a different LLM, and could potentially open up the possibility for testing uncensored models,
(3) Multi-model LLMs for understanding what would normally be shown graphically to a penetration tester is also another avenue for future work.
(4) The RAG datastore used for this study was loaded with data that was largely unstructured. It did, nevertheless, provide insights that were likely to have been useful to the Vulnerability Scan Agent. A more structured approach to loading vulnerability data might improve performance still further and would be worth evaluating. Moreover, the ability of Agents to use the correct Metasploit exploits could be improved significantly if an up-to-date list were provided.
(5) The Metasploitable 2 VM used for this research has vulnerabilities that are relatively easy to exploit, but representative of vulnerabilities that are commonly found by penetration testers. The next step should be to evaluate the performance of LLM against a selection of VMs from Vulnhub, or Hackthebox challenges,
(6) Although a good selection of LLMs were able to be tested during this study, the hardware limitations meant that large LLMs could not be. At the time of writing, we have now secured test time on a supercomputer with 2944 CPU cores, 14x A100 GPUs and 6x L40 GPUs and will be updating our results in due course. In particular we are keen to test the Qwen3:235b and Qwen3.5:122b models.

## REFERENCES


[1] Khalifa Afane, Wenqi Wei, Ying Mao, Junaid Farooq, and Juntao Chen. 2024. Next-Generation Phishing: How LLM Agents Empower Cyber Attackers. In *2024 IEEE International Conference on Big Data (BigData)*. 2558–2567. https://doi.org/10.1109/BigData62323.2024.10825018 ISSN: 2573-2978.

[2] D.M. Anisuzzaman, Jeffrey G. Malins, Paul A. Friedman, and Zachi I. Attia. 2025. Fine-Tuning Large Language Models for Specialized Use Cases. *Mayo Clinic Proceedings: Digital Health* 3, 1 (March 2025), 100184. https://doi.org/10.1016/j.mcpdig.2024.11.005

[3] Siam Shibly Antar. 2025. Pioneering Autonomous Penetration Testing with Large Language Models through Prompt Engineering and Agentic System Design. (Jan. 2025).

[4] Milad Aslaner. 2024. *Cybersecurity Strategies and Best Practices: A comprehensive guide to mastering enterprise cyber defense tactics and techniques* (1 ed.). Packt Publishing Limited, Birmingham.

[5] Stanislas G. Bianou and Rodrigue G. Batogna. 2024. PENTEST-AI, an LLM-Powered Multi-Agents Framework for Penetration Testing Automation Leveraging Mitre Attack. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*. 763–770. https://doi.org/10.1109/CSR61664.2024.10679480

[6] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2024. PentestGPT: An LLM-empowered Automatic Penetration Testing Tool. https://doi.org/10.48550/arXiv.2308.06782 arXiv:2308.06782 [cs].

[7] Luca Gioacchini, Marco Mellia, Idilio Drago, Alexander Delsanto, Giuseppe Siracusano, and Roberto Bifulco. 2024. AutoPenBench: Benchmarking Generative Agents for Penetration Testing. https://doi.org/10.48550/arXiv.2410.03225 arXiv:2410.03225 [cs].

[8] Jonathan Gregory and Qi Liao. 2024. Autonomous Cyberattack with Security-Augmented Generative Artificial Intelligence. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*. 270–275. https://doi.org/10.1109/CSR61664.2024.10679470

[9] Andreas Happe and Jürgen Cito. 2025. Can LLMs Hack Enterprise Networks? Autonomous Assumed Breach Penetration-Testing Active Directory Networks. http://arxiv.org/abs/2502.04227

[10] Andreas Happe, Aaron Kaplan, and Juergen Cito. 2025. LLMs as Hackers: Autonomous Linux Privilege Escalation Attacks. https://doi.org/10.48550/arXiv.2310.11409 arXiv:2310.11409 [cs].

[11] Ismayil Hasanov, Seppo Virtanen, Antti Hakkala, and Jouni Isoaho. 2024. Application of Large Language Models in Cybersecurity: A Systematic Literature Review. *IEEE Access* 12 (2024), 176751–176778. https://doi.org/10.1109/ACCESS.2024.3505983

[12] Isamu Isozaki, Manil Shrestha, Rick Console, and Edward Kim. 2024. Towards Automated Penetration Testing: Introducing LLM Benchmark, Analysis, and Improvements. (2024). https://doi.org/10.48550/ARXIV.2410.17141 Publisher: arXiv Version Number: 4.

[13] Aikaterini Kanta, Iwen Coisel, and Mark Scanlon. 2024. A comprehensive evaluation on the benefits of context based password cracking for digital forensics. *Journal of Information Security and Applications* 84 (2024), 103809. https://doi.org/10.1016/j.jisa.2024.103809

[14] He Kong, Die Hu, Jingguo Ge, Liangxiong Li, Tong Li, and Bingzhen Wu. 2025. VulnBot: Autonomous Penetration Testing for A Multi-Agent Collaborative Framework. https://doi.org/10.48550/arXiv.2501.13411 arXiv:2501.13411 [cs].

[15] Blake E. Strom, Andy Applebaum, Douglas P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. 2020. *MITRE ATT&CK®: Design and Philosophy*. Technical Report. The MITRE Corporation, McLean, VA.

[16] Benlong Wu, Guoqiang Chen, Kejiang Chen, Xiuwei Shang, Jiapeng Han, Yanru He, Weiming Zhang, and Nenghai Yu. 2024. AutoPT: How Far Are We from the End2End Automated Web Penetration Testing? https://doi.org/10.48550/arXiv.2411.01236 arXiv:2411.01236 [cs].