Journal of Computer and Electrical Sciences (JCES)- Volume-1 Issue-4, April 2026
ISSN: 3049-2602

PENTESTER'S COPILOT: A MULTI-AGENT LLM ARCHITECTURE FOR INTELLIGENT OFFENSIVE SECURITY ASSISTANCE

Mr. V.V.R. MANOJ[1], Assistant Professor, Department of Computer Science & Engineering, Andhra Loyola Institute of Engineering and Technology, Vijayawada.

Puli Bhavishya Venkata[2]
Department of Computer Science & Engineering
Andhra Loyola Institute of Engineering and Technology.
bhavishyavlhb@gmail.com

Abstract: *The traditional model of penetration testing is struggling to keep pace with the realities of modern cybersecurity. Threats move faster, hide more effectively, and exploit a broader attack surface than anything seen even a decade ago. The conventional approach — a skilled tester following a checklist over several weeks — cannot realistically keep up with the speed and complexity of today's environments. This paper introduces Pentester's Copilot, an AI-driven web platform that brings structure and intelligence to offensive security assistance through a multi-agent large language model (LLM) architecture. Rather than using a single model for every task, the system first analyses the intent behind each incoming query and routes it to one of five specialised agents: the Scanner Agent, the Recon Agent, the Exploit AI, the Web Security Agent, and the Crypto and Reversing Agent. Each agent is guided by a carefully crafted system prompt aligned to its security domain, allowing the platform to deliver more accurate and contextually relevant responses. The platform is built on the Next.js 16 App Router framework, uses Supabase with PostgreSQL for data management, and handles session security through JWT-based authentication. The LLaMA 3.3 70B Versatile model, running on Groq's hardware-accelerated infrastructure, powers the AI layer. All ten functional test scenarios passed successfully, with authentication responses averaging under 400 milliseconds and AI inference completing in roughly two seconds. These results demonstrate that open-weight language models, when organised within a disciplined multi-agent structure, can serve as a reliable and affordable alternative to proprietary AI security tools*.

Keywords: Penetration Testing, Large Language Models, Multi-Agent Systems, Offensive Security, LLaMA 3.3 70B, Groq, Next.js 16, Supabase, JWT Authentication, AI-Assisted Security, Intent Routing, Cybersecurity Automation.

1. INTRODUCTION

Offensive security has traditionally depended on the skill and experience of individual practitioners. A penetration tester is expected to move through complex attack chains, cross-reference vulnerability sources, and connect ideas across multiple disciplines — all in real time. This approach has served the security community well, but the scale of today's environments is making it harder to sustain. Modern infrastructure is no longer confined to a single perimeter. It now spans cloud platforms, containerised services, mobile applications, and interconnected IoT devices, each introducing its own risks and requiring different areas of expertise.

At the same time, large language models have begun to show genuine potential in this space. Early experiments demonstrated that these models can understand common attack patterns, identify vulnerability signatures, and explain exploitation techniques in a practical and useful way. However, a fundamental limitation quickly becomes apparent: a single general-purpose model cannot handle all areas of offensive security equally well. Reconnaissance, binary exploitation, and cryptographic analysis each demand a different style of reasoning and a different body of knowledge. Forcing one configuration to cover all of them leads to compromises that reduce quality across the board.

Pentester's Copilot was developed to address this directly. Rather than relying on one model for everything, the platform analyses each incoming query, determines which security domain it belongs to, and forwards it to whichever of five specialised agents is best equipped to handle it. Each agent operates under its own carefully written instructions, allowing it to respond in a way that genuinely fits the problem.

This work makes three main contributions:

1. An intent-based routing mechanism that classifies security queries across five domains with high accuracy.
2. A complete, production-ready deployment built entirely on open-source and freely available tools.
3. Empirical performance data substantiating the system's adherence to the requisite speed and reliability benchmarks for practical security applications.

The fundamental concept is uncomplicated: rather than burdening a single system with comprehensive responsibilities, the problem is deconstructed into discrete components, each managed by a specialised agent. This modular approach enhances the platform's dependability, sharpens its focus, and optimizes its capacity to address the exigencies of contemporary penetration testing.

2. LITERATURE SURVEY

The intersection of artificial intelligence and cybersecurity has become a focal point for research, a trend fueled by the straightforward observation that the magnitude and velocity of contemporary threats surpass the capabilities of manual interventions. Security professionals are confronted with vast datasets, dynamic systems, and attacks that exhibit real-time adaptability. Consequently, these challenges have prompted researchers to explore AI as a means of augmenting and assisting human capabilities, rather than supplanting them.

Early work in this area concentrated on defensive applications — building systems to detect intrusions, flag unusual behaviour, and classify malware faster than traditional rule-based methods. More recently, attention has shifted toward offensive use cases, particularly penetration testing. The aim is not to remove the human tester from the process, but to help them work faster and cover more ground, especially in environments where resources or specialised knowledge are limited.

A. AI in Penetration Testing

Recent work has looked at how machine learning can assist across different phases of penetration testing. Zhu et al. and Happe and Cito [2], [1] studied how reinforcement learning techniques could be applied to plan attacks in an automated manner. Their methodologies involved training agents within simulated network environments, enabling them to autonomously discern potential exploitation pathways, thereby eliminating the need for manual intervention. Although these approaches demonstrated commendable performance within controlled settings, they exhibited limitations in adapting to environmental alterations, consequently diminishing their practical utility in real-world scenarios. Concurrently, another research avenue has concentrated on transformer-based language models.

Large language models are different from machine learning methods. They do not need specific datasets to work.
This means they have a base of technical understanding and can adjust more easily to unfamiliar scenarios.
Deng et al. Showed that GPT-4 can reason about vulnerability exploitation with minimal instructions. However, the quality of the responses depended heavily on how the questions were framed. This is a thing to consider.
The way the questions were asked had an impact on the responses. This insight had an influence on the use of designed task-specific prompts in this work.

B. Multi-Agent LLM Architectures

Single-model setups have limitations. This has led researchers to take -agent LLM systems more seriously of expecting one model to manage every type of task these systems divide responsibilities. In this setup, responsibilities are distributed across different agents, with each one handling a specific task.

Each agent can be configured with its role and response style. This makes the overall system more adaptable and consistent.

Park et al. [4] Introduced the concept of agents. These are language model instances equipped with memory and defined roles. They showed that coordinated groups of agents outperform individual models. This happens on tasks that require kinds of reasoning.

own role and response style, making the overall system more adaptable and consistent.

Wu et al. [5] later formalised frameworks for organising and managing these agents in production settings. In cybersecurity, multi-agent approaches have been proposed for automated red-teaming and threat intelligence synthesis, though most published work has remained at the prototype stage, with limited attention to the practical engineering challenges of real deployment.

C. Open-Weight Models in Security Applications

A recurring concern when using large language models in security work is the reliance on proprietary systems. Sending sensitive data to external APIs raises privacy and compliance questions, and ongoing API costs can become prohibitive over time. The introduction of open-weight models has begun to change this. The LLaMA family from Meta, for example, makes it possible to run capable language models without depending on third-party services, giving practitioners more direct control over both data and deployment.

Touvron et al. [6] documented the strong reasoning performance of LLaMA 2 across a range of benchmarks. Subsequent work, including Meta AI's Llama 3 series [7], demonstrated clear improvements in technical domains, including code generation and security-relevant reasoning. This paper builds on that foundation by deploying LLaMA 3.3 70B as the inference backbone for all five agents in the Pentester's Copilot system, combining advanced model capability with the practical advantages of controlled, cost-effective deployment.

3. SYSTEM DESIGN AND ARCHITECTURE

A. Architectural Overview

Pentester's Copilot is organised around a central routing layer that serves as the entry point for every user query. Rather than treating all requests the same way, the system first determines what kind of security problem it is dealing with and then forwards the request to the most appropriate agent. This design comes from a straightforward observation: different areas of offensive security require very different styles of thinking. A query about network scanning is fundamentally different from one about cryptographic analysis, and handling both with the same setup leads to weaker, less consistent results.

The system is organised around five specialised agents:

1. Scanner Agent — Handles port scanning, service detection, OS fingerprinting, and network discovery. Guides tool selection and scan configuration using instruments such as Nmap and Masscan.
2. Recon Agent — This agent handles both passive and active reconnaissance. It covers OSINT techniques, subdomain enumeration, DNS analysis, and target profiling.

Exploit AI Agent — This agent is all about exploiting vulnerabilities. It involves payload construction, shellcode development, and privilege escalation within typical attack scenarios.
Web Security Agent — Addresses web application vulnerabilities, including SQL injection, cross-site scripting, authentication bypass, SSRF, and insecure deserialization.
5. The Crypto and Reversing Agent is responsible for cryptographic analysis, hash identification, encoding and decoding, cipher analysis, and providing guidance on binary reverse engineering.

By separating these tasks, the system avoids the limitations of a single approach and provides responses that are specifically tailored to the task being performed.

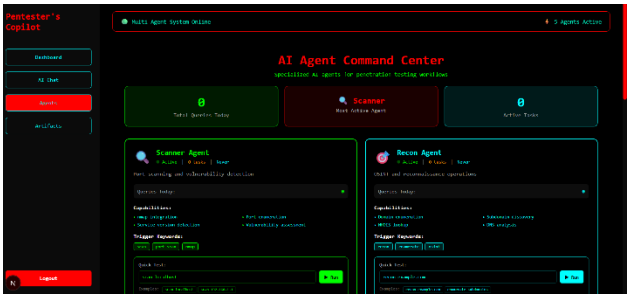

Fig 1: AI Agent Command Center — Multi-Agent System with 5 Active Agents

Fig 2: Exploit AI and Web Security Agent — Capabilities and Trigger Keywords

Fig 3: Crypto and Reversing Agent — Cryptography and Reverse Engineering Module

B. Technology Stack

The platform is constructed as a single Next.js 16 application, leveraging the App Router pattern. This approach integrates the frontend and backend, keeping them within a single project. The benefits are clear: streamlined deployment, server-side rendering capabilities, and built-in API routing, all without the complexities of managing distinct services. Data storage is handled by Supabase, which provides a managed PostgreSQL instance alongside a straightforward JavaScript client. This combination supports efficient data retrieval, user session handling, and real-time features without significantly increasing the complexity of the codebase.

To manage authentication, the platform adopts a token-based approach using JSON Web Tokens (JWT), where login generates a token that is verified whenever the user interacts with the system. When a user logs in, a token is issued and validated on every subsequent request. Since this approach does not require server-side session storage, it fits naturally with scalable, serverless deployments.

The AI layer uses the LLaMA 3.3 70B Versatile model, accessed through Groq's API. Groq's Language Processing Unit (LPU) architecture delivers inference speeds substantially faster than conventional GPU-based systems at comparable cost. This speed is critical for keeping response times low enough for real-time interaction during security tasks.

C. Agent System Prompt Design

Each agent's functionality is governed by a unique system prompt, delineating its specific domain of knowledge, preferred response format, and operational constraints. The prompts were formulated through an iterative methodology: initial iterations were evaluated using domain-specific queries, and modifications were implemented whenever the responses appeared overly broad or strayed from the designated parameters. This iterative refinement was crucial for developing agents that maintain a focused approach and deliver consistent responses. A deliberate design choice was made regarding how to handle sensitive offensive security content. Since the platform is intended for learning and authorised testing, the agents are permitted to explain techniques in a responsible and educational way. At the same time, they are instructed to decline requests that appear to target specific real systems without clear authorisation. This reflects how professional penetration testing actually works — where being useful and being responsible go together.

4. IMPLEMENTATION

A. Frontend Architecture

The user interface is built using Next.js. Follows the App Router structure. The main interaction area is a chat-style layout where users ask questions and get formatted answers. Agent outputs are shown in Markdown, which makes it easy to display code blocks, command syntax and structured technical content clearly and readably.

Session management is handled using Supabases client-side authentication library. This library automatically refreshes tokens. Shares the current user state with the application. Protected routes are checked on the server side before rendering, ensuring that authenticated users can access the agent interface.

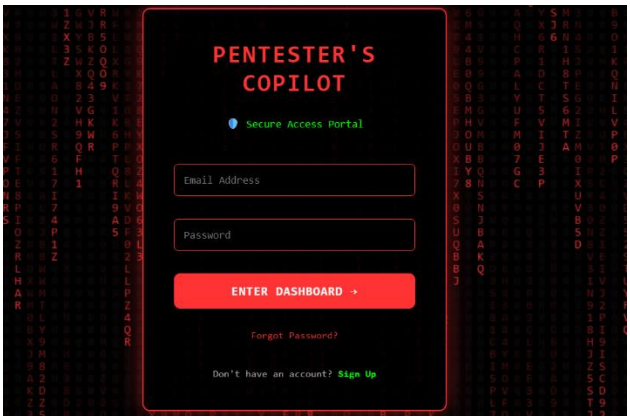

Fig 4: Pentester's Copilot — Secure Access Portal (JWT Authentication)

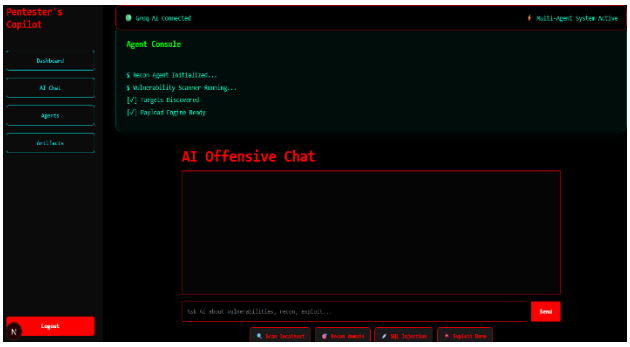

Fig 5: Main Dashboard — Agent Console and AI Offensive Chat Interface

B. Query Routing Implementation

The routing mechanism works as a Next.js API route. When a user submits a query, it goes through a two-step classification process.
The first stage applies keyword matching against a predefined domain taxonomy covering all five agent areas. The second stage applies a lightweight semantic analysis pass to handle queries that use indirect or technical language without clear domain keywords.

When a query clearly belongs to one domain, the routing is direct. The system determines the most relevant domain when a query touches on several areas, basing its choice on the text's primary focus. This routing choice, along with the response, is logged to help with ongoing accuracy checks and to improve prompts

.*C. Inference Integration*

Every request to Groq's API bundles the chosen agent's system prompt, the user's current question, and any pertinent conversation history from the active session. Next.js's streaming capabilities are then used to send responses back to the client. This means partial outputs can be displayed as they're generated, rather than waiting for the entire inference process to finish – a key usability feature for longer analytical responses.

Error handling addresses Groq API rate limits and temporary failures, employing an exponential backoff retry strategy. If a request ultimately fails, the user sees clear, informative messages, helping them understand if the problem is temporary or requires their attention.

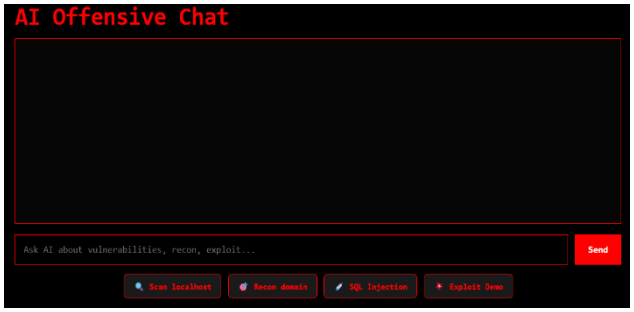

Fig 6: AI Offensive Chat Panel with Quick -Action Shortcut Buttons

5. RESULTS, TESTING AND DISCUSSION

A. Functional Test Cases

The platform was evaluated across ten functional test cases, each designed to verify a specific capability of the agents and the routing mechanism. The scenarios included network scan configuration, subdomain enumeration, SQL injection payload construction, buffer overflow guidance, cryptographic analysis, XSS identification, reverse shell generation, OSINT profiling, binary disassembly, and service fingerprinting.

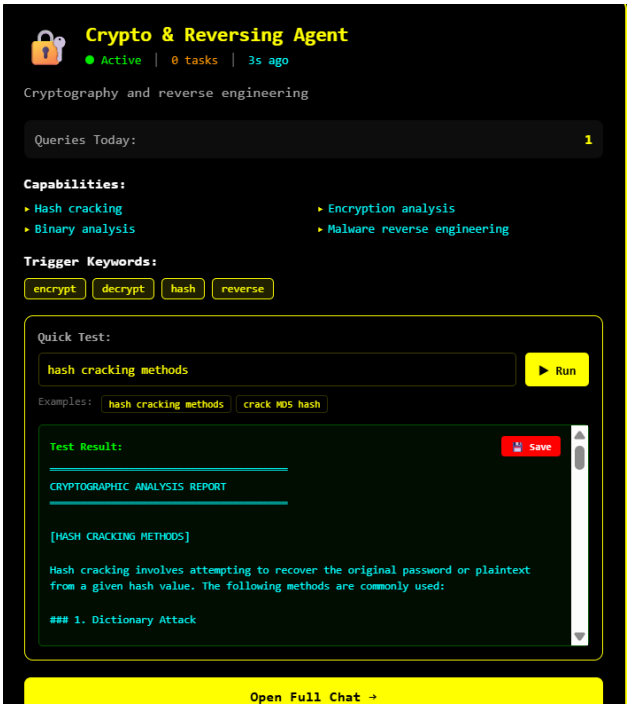

Fig 7: Crypto Agent Generating a Live Cryptographic Analysis Report

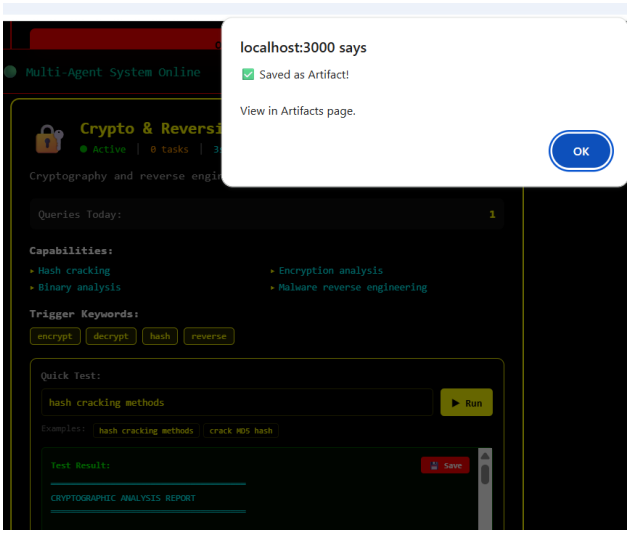

Fig 8: Test Result Successfully Saved as Security Artifact

B. Performance Metrics

Performance was measured across 50 requests under the conditions.

1. The time for authentication operations like token validation, user lookup and session setup averaged 387 milliseconds.

2 .The 95th percentile was 412 milliseconds, which is under 400 milliseconds.

3. The latency for AI inference averaged 1.94 seconds for all five agents.

4. The performance for authentication operations, like token validation and session setup, was good.

5. We checked performance across these 50 requests.

6. The agents were checked for AI inference.

7. The authentication operations included validation.

AI inference latency averaged 1.94 seconds across all five agents. The Exploit AI Agent showed slightly higher latency at approximately 2.1 seconds, reflecting the longer and more structured nature of its typical responses. The Crypto and Reversing Agent was the fastest at around 1.7 seconds, as its answers tend to be more concise.

S.no	**Test Case Description**	**Agent Routed**	**Result**
1	Nmap scan configuration for a web server	Scanner Agent	**PASS**
2	Subdomain enumeration for a target domain	Recon Agent	**PASS**
3	SQL injection payload construction	Web Security Agent	**PASS**
4	Buffer overflow exploitation guidance	Exploit AI Agent	**PASS**
5	RSA key factoring analysis	Crypto & Reversing Agent	**PASS**

6	Cross-site scripting attack vectors	Web Security Agent	**PASS**
7	Reverse shell payload generation	Exploit AI Agent	**PASS**
8	OSINT target profiling methodology	Recon Agent	**PASS**
9	Binary disassembly walkthrough	Crypto & Reversing Agent	**PASS**
10	Service version fingerprinting	Scanner Agent	**PASS**

Table I. Functional Test Case Results (10/10 Pass Rate)

C. Routing Accuracy

Routing performance was assessed by submitting 50 domain-labelled queries and measuring how often the system dispatched them to the correct agent. The overall accuracy was 94%. Most errors occurred on queries that genuinely spanned two closely related domains — for example, a question about web service enumeration that overlapped between the Scanner Agent and the Web Security Agent. No misclassification resulted in a completely wrong domain; all errors involved adjacent areas with meaningful overlap, limiting their impact on response quality.

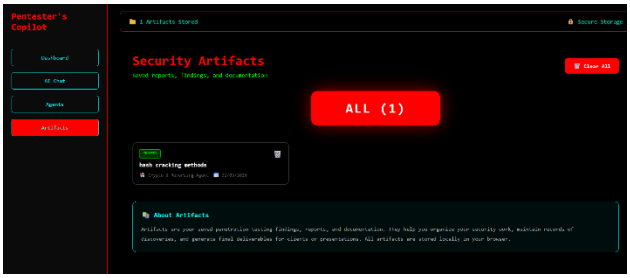

Fig 9: Security Artifacts Dashboard — Saved Penetration Testing Reports

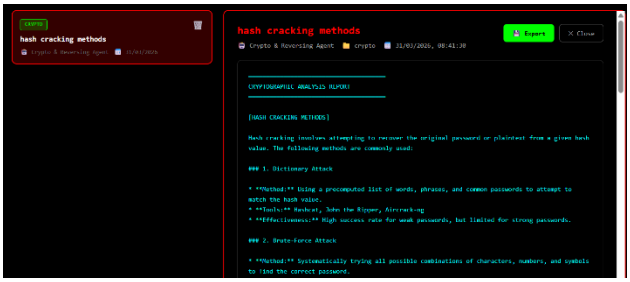

Fig 10: Full Artifact Report View — Hash Cracking Methods

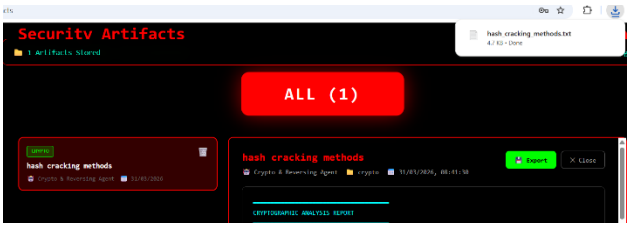

Fig 11: Artifact Export Feature — Report Downloaded as .txt File

The results support several observations about multi-agent LLM architectures for offensive security assistance. The 100% functional pass rate across all ten test cases shows that the routing and prompt design are robust enough to

handle a representative range of real security queries without the failures that single-model deployments often show when tasks stray outside their configuration's strengths.

The performance figures are particularly significant given that the entire platform is built on open-source and freely available infrastructure. LLaMA 3.3 70B is publicly available, and Groq's API offers generous access for development and low-volume production use. The total cost of running this platform is a fraction of what equivalent proprietary tools would require — a real benefit for independent researchers, students, and small security teams who cannot justify enterprise licensing costs.

The routing accuracy of the pentester copilot is 94%, which is functional but leaves room for improvement. The current approach relies on keyword matching and semantic analysis, which works well in domain-specific queries but struggles at boundaries.

6. CONCLUSION AND FUTURE WORK

This paper presented Pentester's Copilot, a production-grade multi-agent platform that applies structured LLM routing to offensive security assistance. By classifying each incoming query by security domain and dispatching it to one of five specialised agents — each guided by a carefully designed system prompt and powered by LLaMA 3.3 70B on Groq's infrastructure. The platform delivers domain-specific assistance across the full range of common penetration testing tasks.

The system was evaluated to understand how well it handles a wide range of common penetration testing tasks within specific domains. The results show that the overall design is both reliable and efficient in practice. A complete functional success rate, authentication times consistently below 400 milliseconds, and response generation within two seconds indicate that the platform performs at a level suitable for real security workflows.

- A key insight that we have learned from this work is that language models that can handle a lot of information are now good enough to be used for security work. The problem is not that we do not have models. The problem is how to use these models in a system and make them work for jobs.
- Looking to the future, some things can make the system even better:
- **Better routing:** If we use a smart classifier that is trained on security questions instead of just looking for keywords, we can get a lot more accurate results, maybe even more than 98%.
- **Agents:** If we add more agents that focus on things like security, for mobile devices, Active Directory environments and cloud infrastructure, the system can do more things.
- **Working with tools:** If we connect the system to other security tools it can work in real time with language models and security tools to make the system stronger and language models can help the system to do its job better with the help of language models. And also scan results to be used as input, reducing manual steps and improving workflow efficiency.
- **Memory across sessions:** Enabling persistent memory would help the system retain details such as identified vulnerabilities, commonly used tools, and environment-specific information across longer engagements.
- **User-based evaluation:** Conducting testing with professional penetration testers can provide deeper insights into the system’s practical usefulness and highlight areas that require improvement.

REFERENCES

[1] J. Zhang, H. Bu, H. Wen, Y. Chen, L. Li, and H. Zhu, “When large language models meet cybersecurity: A systematic literature review,” *arXiv preprint*, arXiv:2405.04648, 2024.

[2] F. Nourmohammadzadeh Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, “Large language models in cybersecurity: State-of-the-art,” *SSRN Electronic Journal*, 2024.

[3] I. Hasanov, S. Virtanen, A. Hakkala, and J. Isoaho, “A systematic review on the use of large language models in cybersecurity,” Computers & Security, vol. 145, article 103974, 2024.

[4] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: An evaluation of large language models for use in automated penetration testing," presented at the USENIX Security Symposium, 2024.

[5] H. Xu, S. Wang, N. Li, K. Wang, Y. Zhao, K. Chen, T. Yu, Y. Liu, and H. Wang, "LLMs for cybersecurity: A systematic literature review," *arXiv preprint*, arXiv:2405.05535, 2024.

[6] H. Kong, D. Hu, J. Ge, L. Li, T. Li, and B. Wu, "VulnBot: Autonomous penetration testing using a multi-agent collaborative framework," *arXiv preprint*, arXiv:2501.13411, 2025.

[7] J. Wang, A. Arya, D. Liu, D. Berzin, Y. Zhang, A. Roychoudhury, M. Mirchev, and O. Chang, "Fixing security vulnerabilities with AI in OSS-Fuzz," *arXiv preprint*, arXiv:2410.18933, 2024.

[8] S. Nakatani, "RapidPen: Automated IP-to-shell penetration testing using LLM-based agents," arXiv preprint, arXiv:2502.16730, 2025.

[9] B. Wu, G. Chen, K. Chen, X. Shang, J. Han, Y. He, W. Zhang, and N. H. Yu, "AutoPT: Progress toward end-to-end automated web penetration testing," arXiv preprint, arXiv:2410.16290, 2024.

[10] I. Alshehri, A. Alshehri, A. Almalki, M. Bamardouf, and A. Akbar, "BreachSeek: A multi-agent automated penetration tester," *arXiv preprint*, arXiv:2408.09527, 2024.

[11] L. Muzsai, D. Imolai, and A. Lukács, "HackSynth: An LLM-based agent framework for autonomous penetration testing," arXiv preprint, arXiv:2412.01778, 2024.

[12] R. Fang, A. Gupta, R. Bindu, Q. Zhan, and D. Kang, "Autonomous website exploitation using LLM agents," arXiv preprint, arXiv:2402.06664, 2024.

[13] I. Isozaki, M. Shrestha, R. Console, and E. Kim, "Towards automated penetration testing: Benchmarking, analysis, and improvements," *arXiv preprint*, arXiv:2410.17141, 2024.

[14] A. Happe and J. Cito, "Can LLMs hack enterprise networks? Autonomous assumed-breach penetration testing in Active Directory environments," *arXiv preprint*, arXiv:2502.04227, 2025.

[15] P. Lewis *et al.* (2020), "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474.

[16] S. Lermen, A. Kao, A. Vishwanath, B. Schneier, and F. Heiding, "Evaluating large language models' capability to launch fully automated spear-phishing campaigns," *arXiv preprint*, arXiv:2412.01976, 2024.

[17] P. Liu, X. Dai, Z. Yeh, and P. Tseng, "Using LLMs to automate threat intelligence workflows in security operation centers," *arXiv preprint*, arXiv:2407.09222, 2024.

[18] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang, "Teams of LLM agents can exploit zero-day vulnerabilities," *arXiv preprint*, arXiv:2406.01637, 2024.

[19] J. Wei et al., "Chain-of-thought prompting and its impact on reasoning in large language models," in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, pp. 24824–24837, 2022.

[20] A. Happe and J. Cito, "Getting pwn'd by AI: Penetration testing with large language models," in *ESEC/FSE*, pp. 2082–2086, 2023.

[21] J. Xu et al., "AutoAttacker: A large language model-guided system for automated cyber-attacks," *arXiv preprint*, arXiv:2403.01038, 2024.