Multi-Agent System for Automated Red Teaming Workflows: Methodology and Architecture

Ioannis Pastellas
Data & AI Systems Group
UBITECH Ltd
Limassol, Cyprus
0000-0002-1193-6280

Sophia Karagiorgou
Data & AI Systems Group
UBITECH Ltd
Limassol, Cyprus
0000-0002-1099-8463

Evangelos Kafantaris
Data & AI Systems Group
UBITECH Ltd
Limassol, Cyprus
0000-0001-9599-7656

***Abstract*—Manual penetration testing can no longer keep pace with modern attack surfaces and AI-accelerated adversarial threats. To this end, we propose an automated red teaming methodology structured across three phases: Reconnaissance, Execution, and Post-Execution in alignment with the National Institute of Standards and Technology Cybersecurity Framework 2.0 and the European Cybersecurity Skills Framework to enable adoption within existing security organizations. Our methodology is implemented via a hierarchical multi-agent architecture, where a supervisor agent coordinates specialized sub-agents under bounded execution contexts. Evaluation occurs in a dockerized environment using phase-specific verifiable scoring functions, with experiments utilizing both proprietary and open-source LLMs, benchmarked against a single-agent baseline. Results show consistent performance gains with the multi-agent architecture over the single-agent architecture with the execution phase representing a bottleneck for open-source models. The work establishes a foundation for scalable, standards-aligned deployment of automated red teaming workflows in operational cybersecurity settings.**

***Index Terms*—automated red teaming, multi-agent systems, penetration testing, security orchestration**

I. Introduction

The rapid expansion of digital infrastructures and the increasing complexity of modern attack surfaces have placed significant pressure on organizations to continuously evaluate and strengthen their cybersecurity posture. This challenge has been further intensified by the emergence of Generative AI (GenAI) capabilities in adversarial workflows, enabling attackers to scale reconnaissance, automate exploit development, and coordinate campaigns with unprecedented speed. Under these conditions, traditional penetration testing which are typically periodic, manual, and resource-intensive can no longer provide sufficient coverage or responsiveness.

These limitations create a need for automated and scalable penetration testing approaches capable of persistently assessing organizational exposure. However, increased automation must be balanced with governance, safety and organizational coordination ensuring that automated security operations are carefully audited and controlled in alignment with established risk management practices. A key barrier in current practices is the fragmented implementation of respective paradigms, which often results in isolated tooling without a coherent operational workflow.

To address this gap, we design an end-to-end methodology for the control and automation of the penetration testing lifecycle structured in three distinct phases informed by the four-phase process of NIST SP 800-115 [1]. In NIST SP 800-115, penetration testing activities are organized in Planning, Discovery, Attack, and Reporting phases. In our methodology, the activities of the Planning phase are allocated to the orchestration layer which also monitors the entire testing lifecycle. The remaining three phases are organized in a serial process flow with Discovery corresponding to our Reconnaissance phase, Attack corresponding to our Execution phase and Reporting corresponding to our Post-Execution phase. During the specification of our phases we prioritized the grouping of target functions in a manner that would allow the effective compartmentalization of target capabilities while also taking into consideration alternative approaches of penetration testing practices [2].

To enable adoption within existing security organizations, we align this methodology with widely adopted cybersecurity frameworks. We align the lifecycle and governance model with the NIST Cybersecurity Framework 2.0 [3], which informs the lifecycle structure, risk-management integration, and control objectives, and draw operational context from ENISA reports [4] and the European Cybersecurity Skills Framework (ECSF) [5], [6] for the threat landscape, regulatory expectations, and role mapping. This dual alignment lowers adoption barriers by letting organizations relate automated testing to existing governance processes and personnel responsibilities.

We implement our methodology via a hierarchical multi-agent architecture in which a supervisory agent coordinates specialized LLM-based sub-agents responsible for reconnaissance, task execution, and post-execution analysis [7]. This modular design enables adaptive task orchestration, controlled execution contexts, and efficient resource management [8]. The goal is to introduce a methodology and an implementation

"This work has received funding from the Digital Europe DETANGLE project under Grant Agreement No. 101249446, the Digital Europe SONIC Project under Grant Agreement No. 101168486 and the Horizon Europe SecQDevOps Project under Grant Agreement No. 101225776. Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the European Cybersecurity Industrial, Technology and Research Competence Centre. Neither the European Union nor the European Cybersecurity Competence Centre can be held responsible for them."

system that leads to improved performance in the completion of designated tasks, stability of its outputs across multiple deployments and traceability of errors.

These design goals are based on a growing body of research that focuses on the utilization of LLM-based systems for offensive testing. PentestGPT [9] operates as a single-agent and decomposes a given task into reasoning, generation, and parsing modules. hackingBuddyGPT [10] automates Linux privilege escalation with a single agent while single GPT-4 agents have been utilized for the exploitation of one-day vulnerabilities given a CVE description [11]. More recent systems distribute the workflow across cooperating agents: PentestAgent [12] adds retrieval over reconnaissance, search, planning, and execution agents, VulnBot [13] coordinates a penetration testing team of agents via a task graph, and CAI [14] offers reusable agentic patterns. We therefore see a consistent pattern regarding the potential of agentic task decomposition that we want to expand upon in the following three axis (i) the alignment of each process phase with NIST CSF 2.0 functions and ECSF roles, giving a lifecycle governance mapping absent from prior systems, (ii) the evaluation of each phase with targeted and verifiable scoring functions and the quantification of output stability across multiple iterations (iii) the conduction of controlled single-agent versus multi-agent comparisons on identical proprietary and locally hosted open-source foundational models.

The rest of the paper is organized as follows: Section II. Methods describes the design of our 3-Phase Methodology, its associated Hierarchical Multi-Agent Architecture, implemented Safety and Scope Controls, and the Experimental and Evaluation Setup followed in our study. Section III. Results & Discussion presents in detail experimental results and benchmarking comparisons, while also highlighting key points of interest and opportunities for future work. Finally, Section IV. Conclusion summarizes the work presented in the manuscript.

II. Methods

A. Design of the 3-Phased Methodology

The proposed methodology defines a three-phase lifecycle for the development and deployment of automated penetration testing routines, structured to ensure alignment with established cybersecurity governance and operational practices. Consequently, each phase is mapped to two functions of the NIST CSF 2.0, and respective cybersecurity roles as defined by the European Cybersecurity Skills Framework (ECSF) of ENISA. This mapping reinforces the traceability of the activities performed by the Agentic Red Team ensuring that cybersecurity teams can directly adapt the Agentic System to their operations by leveraging existing and widely adopted frameworks and team compositions. To this end, we conduct our mapping in a manner that aims to specify for each penetration testing phase: (i) which of the NIST CSF functions have the closest alignment with phase specific outcomes and (ii) which of the ECSF roles would be ideal candidates to audit and optimize its operations.

Consequently, we map the Reconnaissance phase to the Identify and Protect functions of the NIST framework through the involvement of Cyber Threat Intelligence Specialists and Cyber Security Architects. The Execution phase is mapped to the NIST Detect and Respond functions, primarily utilizing the expertise of Penetration Testers and Cyber Incident Responders. Finally, the Post-Execution phase is mapped to the Recover and Govern functions, aligned with Cybersecurity Auditors and Cyber Compliance Officers.

The structure of our methodology provides the direct foundations for a multi-agent architecture. In particular, each of the three phases is allocated to a specialized agent who operates under the control of a supervisor agent overseeing the entire process [15]. This modular methodological adaptation enables separation between iterative components, such as repeated scanning or exploitation attempts (handled at the level of phase specific agents), and persistent components, where findings from earlier stages inform subsequent actions (handled by the supervisor). By decomposing responsibilities across coordinated specialized agents under supervisory control, the system provides a persistent, role-structured assessment capability that functionally parallels a coordinated security testing team while maintaining centralized human supervision and governance. Consequently, our design incorporates human-in-the-loop oversight, strict execution boundaries, and safety controls to ensure trustworthy operation within authorized environments. The combination of these capabilities allows a team of cybersecurity professionals to leverage the direct support of a scalable Agentic AI Red Team, that is, the hierarchical multi-agent system introduced in Section II-B, which autonomously performs red-teaming workflows under human supervision, enabling the scalable implementation of auditable penetration testing.

B. Hierarchical Multi-Agent Architecture

We implement our methodology as a two-tier hierarchical multi-agent system using the subagents-as-tools pattern built on top of LangChain [16] and its underlying graph execution engine. Unlike monolithic single-agent designs, this architecture decomposes the penetration testing workflow into specialised sub-agents, each governed by a dedicated system prompt and an independent step budget, while a top-level supervisor coordinates task delegation and results aggregation. The subagents-as-tools pattern wraps each sub-agent as a callable tool, making it invocable by the supervisor through the standard LangChain tool-calling interface. This design ensures the following properties: (i) Coordination capabilities from the supervisor who is capable of selecting which sub-agent to invoke and what context to pass, treating delegation as a structured tool call and (ii) controlled operation at the level of each sub-agent who operates within its own bounded execution context (setup prompt, tools, step limit), preventing unbounded resource consumption.

1) Agentic Interactions: The interactions between the Agents of our architecture can be understood within the context of a two-tier hierarchy. The first tier is comprised

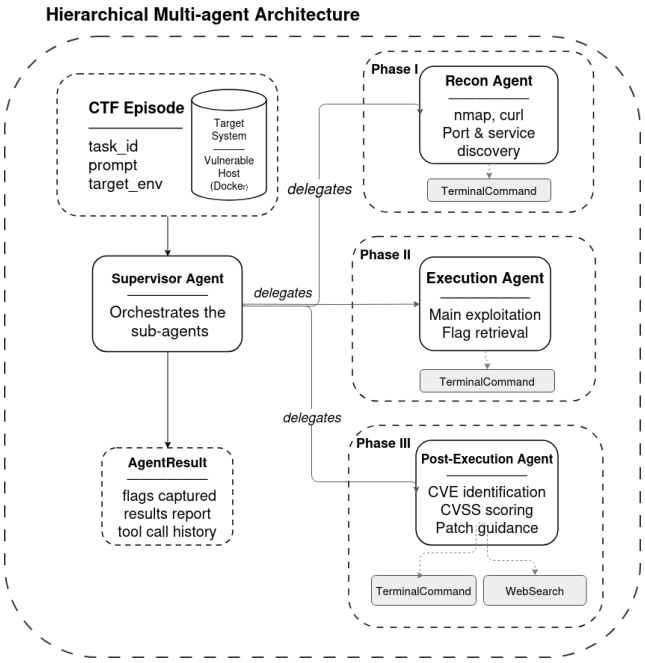

Fig. 1: Hierarchical Multi-Agent Architecture using the subagents-as-tools pattern. Grey nodes represent tools. Rectangular nodes represent agents with their governing prompts. Dashed rectangles denote the phases of our methodology and the workflow results made available via the Supervisor Agent.

of the supervisor and the tool-wrapped entry points for each sub-agent. Subsequently, for each sub-agent invocation, a second inner tier is instantiated containing the sub-agent's own reasoning loop calibrated towards the utilization of each associated tools. The resulting Multi-Agent system consists of the following components:

1) Supervisor Agent: the top-level coordinator that receives the user prompt, decides which sub-agents to invoke, aggregates their outputs and synthesises the final answer.
2) Reconnaissance Agent: focusing on port scanning and service discovery.
3) Execution Agent: focusing on exploiting detected vulnerabilities and conducting post-exploitation processes
4) Post-Execution Agent: focusing on documenting the penetration testing process in a manner that enhances human supervision and subsequent cybersecurity operations.

The architecture of the Hierarchical Multi Agent System is illustrated in Fig. 1. The core of the system is the Supervisor Agent, which orchestrates specialized sub-agents to interact with a vulnerable host residing in a Docker container in CTF (Capture-The-Flag) setting. The operational flow is divided into three sequential phases.

In Phase I, the Recon Agent conducts initial discovery using network mapping and data transfer tools to identify open ports and active services. During Phase II, the Execution Agent performs the primary exploitation tasks and manages the retrieval of flags from the target system. Phase III involves the Post-Execution Agent, which facilitates vulnerability assessment by identifying common vulnerabilities and exposures, determining the respective common vulnerability scores, and generating associated patch guidance. Throughout these phases, the agents utilize terminal commands (bash) and external search capabilities to interact with the environment. The process concludes with the generation of the result of our system, which contains the captured flags, a detailed results report, and the complete history of tool calls. The keyword TerminalCommand refers to a bash command, and the keyword WebSearch to a basic web search tool capability.

C. Safety and Scope Controls

Automated penetration testing agents must be carefully constrained to prevent unintended interactions outside the designated target environment. To achieve this, the architecture implements the following sequential control processes:

1) First, a per-iteration IP target list is maintained to prevent the system from attacking hosts beyond the designated targets and to control the scope of the attack
2) Second, all tool calls are wrapped in error-handling middleware that captures detected errors and generates tool-specific message responses. These responses are returned to the agent to kickstart an adaptation process at the level of each sub-agent.
3) Third, a step budget is allocated for each agent to ensure that unbounded execution is prevented if the adaptation process fails after a designated number of steps.
4) Finally, prompt-level constraints are implemented through explicit scope rules in the prompts delivered by the Supervisor Agent to the sub-agents regarding the targets to be engaged and the actions to be undertaken.

To evaluation the performance of our architecture across different axes of the penetration testing workflow we design the following experimental setup.

D. Experimental Setup

To implement our benchmarking framework we developed an evaluation pipeline that interfaces the autonomous agents with a sandboxed, vulnerable target environment. The resulting system is designed to measure at each experimental iteration the performance across all three phases of our workflow (reconnaissance, execution, post-execution). The target for our experiments is a containerized instance of **Metasploitable2** [17], a Linux distribution purposefully engineered with a high-density attack surface. We utilize a Docker-based orchestration to manage the target infrastructure to ensure we can have rapid programmatic resets of the environment between trials and enforce strict network isolation. Furthermore, the lightweight nature of Docker allows for scalable and parallel evaluations streamlining the benchmarking of stochastic Large Language Model (LLM) behaviors.

To this end, we utilize a mixture of open-source and proprietary architectures. From the open-source domain we deploy Qwen3-8B [18], Gemma4-E4B (4.5B) [19], and the lightweight Qwen3-4B [18], all selected for their tool-calling capabilities. As a proprietary baseline we test GPT-5-mini [20],

a high-reasoning model from OpenAI.Open-source models are hosted on-premise on local hardware using Ollama [21]. Per LLM-model we benchmark the performance of our architecture by comparing its results to a baseline single-agent architecture using the same model. The process is repeated for a total of 10 iterations per experimental setup to measure the stability of each system's output profile.

In detail, the single-agent baseline implements a ReAct-style reasoning loop in which a single agent alternates between generating an action and observing its results in the target environment. The agent has access to a single tool that executes shell commands inside the isolated lab network and returns their standard output and it receives a single, self-contained prompt that encodes the complete workflow methodology as an ordered sequence of 3 mandatory phases. No additional input is provided between phases, the model must carry all accumulated findings in its context window as it advances through the workflow. The following evaluation processes are designed to grade the success of each subtasks of our penetration testing workflow in alignment with the 3-phases of our methodology:

1) Reconnaissance Validation: The system validates this phase by parsing the agent's terminal output for nmap results. A regex-based heuristic extracts port/protocol pairs, using a strict pattern for standard output and a looser pattern for Markdown tables or free-form summaries. We define n as the number of distinct ports recovered and M as the set of expected services missing from the evidence. With a minimum requirement of $n_{\min} = 3$, the recon score is assigned as: 0 when $n = 0$; 0.5 when $0 < n < n_{\min}$; 0.75 when $n \geq n_{\min}$ but $M \neq \emptyset$; and 1 when both $n \geq n_{\min}$ and $M = \emptyset$.

2) Execution Verification: We employ a rigid verification mechanism based on proof-token retrieval to confirm successful exploitation. The agent must retrieve a unique cryptographic string (`FLAG{...}`) stored in a restricted directory on the target Metasploitable2 instance, accessible only via a shell exploit or privilege escalation. These tokens are generated randomly upon container setup. The engine performs a direct comparison between the agent's reported token and the ground truth. A successful match is required to prevent "hallucinated" progress where an LLM might claim a compromise without gaining functional control.

3) Post-Execution Quality: To assess the technical depth of the remediation reports, we utilize an *LLM-as-a-judge* (GPT-4o-mini). The judge evaluates reports against a five-point integer rubric: (0) no report provided, (1) generic security advice only, (2) correct vulnerability identification with vague fixes, (3) accurate CVE citation and clear fix steps, and (4) inclusion of actionable remediation and official references. The resulting integer is divided by 4 to produce a final normalized score.

III. Results & Discussion

A. Single-Agent and Multi-Agent Benchmarking Results

Our CTF-style evaluation in a controlled Dockerized Metasploitable2 environment covers: (i) service discovery, (ii) exploitation with privilege verification and proof-token retrieval, and (iii) post-execution reporting. Table I reports, baseline results based on the single-agent architecture versus the proposed multi-agent architecture on a per phase basis for each evaluated LLM-model. Each table entry presents the average value in a normalized range of 0 to 1 across a total of 10 iterations per experiment, with each experiment consisting of a complete reconnaissance, execution and post-execution cycle. Finally, Fig. 2 provides an analytical breakdown of the score distribution displaying side by side the single and multi-agent configuration results. For each phase, score value is associated with positioning on the x-axis and the number of iterations reaching the associated score value is displayed on the y-axis. Upon reviewing the benchmarking results the following observations can be made on a per-phase basis.

Reconnaissance: For this phase a transition from a single-agent to a multi-agent consistently improves performance stability across all models. GPT-5-mini receives a minor performance increase moving from an almost perfect profile in the single-agent case to a perfect task completion profile in the multi-agent case. Both Qwen-8B and Gemma4 E4B display significant performance improvements when moving to a multi-agent architecture achieving a perfect output profile matching that of GPT-5-mini. Finally, the most significant performance increase is noted for the Qwen-4B model which moves from a complete failure profile to a 80% success rate.

Execution: This phase poses the greatest challenge with only the GPT-5-mini model successfully completing the phase and the open-source models failing in both single and multi-agent configurations. Regarding GPT-5-mini an improvement in the stability of its performance is observed when moving to our multi-agent architecture resulting in a perfect output profile.

Post-execution: For this phase the effects of moving from single-agent to multi-agent architecture differ significantly based on the underlying LLM-model. For GPT-5-mini performance increases when using the multi-agent architecture leading to a consistently perfect score of 1.0. Qwen3-8B displays the most noticeable increase with its performance moving from a wide distribution of scores to a more right tailed distribution with 70% optimal score. For models with smaller number of parameters (Gemma4-E4B and Qwen3-4B) we observe an inconsistency in performance changes. For Gemma4-E4B moving from a single-agent to a multi-agent architecture introduces minor changes in the output profile which actually results in a minor decrease of the average score from 0.35 to 0.30. In contrast, for Qwen3-4B an increase in performance is observed moving from an average score of 0.13 to 0.45.

B. Discussion & Future Work

Our multi-agent architecture enables compartmentalized evaluation of each sub-agent, allowing performance bottlenecks to be localized and calibrations to be applied either at the sub-agent level or at the supervisor–sub-agent interface.

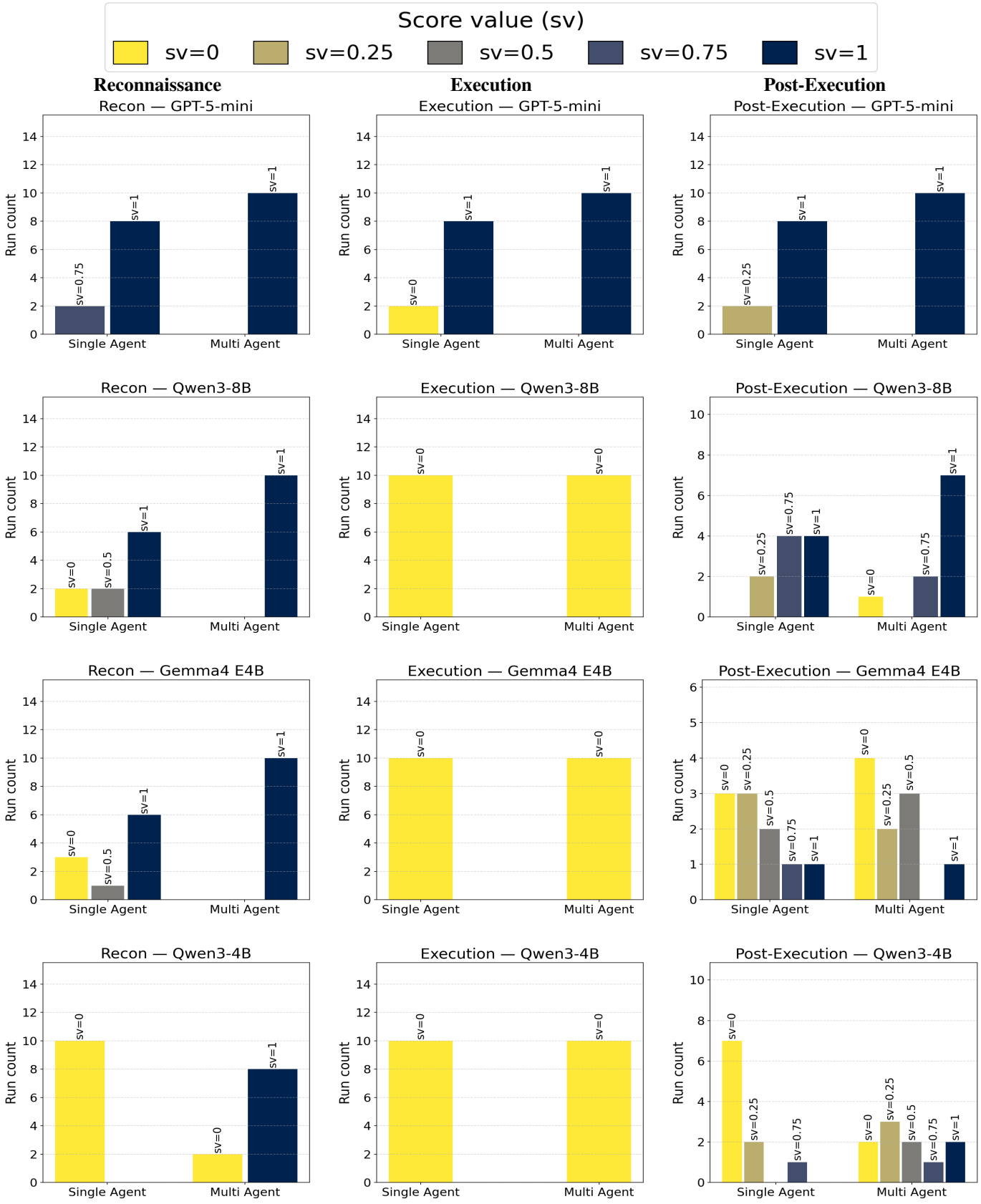

Fig. 2: Score distribution per evaluator comparing single-agent versus multi-agent configurations across the three phases of Reconnaissance, Execution, Post-Execution for each model. For each plot, columns indicate the number of iterations for which a phase-specific score was achieved.

TABLE I: Average per-phase score (0–1) over 10 iterations for Single-Agent (S) and Multi-Agent (M) architectures.

	Recon.		Execution		Post-Exec.	
LLM	S	M	S	M	S	M
GPT-5-mini	0.95	1.00	0.80	1.00	1.00	1.00
Qwen3-8B	0.70	1.00	0.00	0.00	0.60	0.80
Gemma4-E4B	0.65	1.00	0.00	0.00	0.35	0.30
Qwen3-4B	0.00	0.80	0.00	0.00	0.13	0.45

Moving from a single-agent to our multi-agent architecture yields three observations: (i) the multi-agent architecture is more stable across iterations and as Qwen3-4B shows in reconnaissance, can lift lightweight models past tasks they previously failed, (ii) execution is the most demanding phase, requiring long-horizon planning, robust interpretation of noisy fingerprints, and recovery after failed attempts, and (iii) post-execution reporting is less tool-intensive and can succeed even when exploitation fails, though the gain for low-parameter models is far smaller than in reconnaissance.

Moving forward, we aim to expand this study to investigate: (i) alternate phase-evaluation mechanisms to test the persistence of the observed patterns, (ii) more LLMs, including different models per sub-agent, (iii) decomposing the execution phase into smaller tasks with additional tools to make open-source models viable there, and (iv) monitoring token count and energy use to optimize resource utilization.

IV. Conclusion

In this paper, we presented an end-to-end methodology for automated red teaming, implemented via a hierarchical multi-agent architecture. By aligning the penetration testing lifecycle with the NIST CSF 2.0 and the ECSF, we established a 3-phase methodology that enables organizations to integrate automated security assessments into existing operational roles and risk management processes. Our experimental evaluation demonstrates consistent improvements in overall performance and stability when moving from a single-agent to our multi-agent architecture. A current bottleneck to be addressed is the failure of locally hosted open-source models to successfully complete the execution phase of the workflow. To this end, we aim to expand our study with additional evaluation mechanisms, the deployment of multi-agent systems utilizing different LLM-models at the sub-agent level and the potential decomposition of the execution phase into smaller tasks. Our current methodology and architecture lays the foundation towards the scalable deployment of Automated Red Teaming workflows in an effort to empower cybersecurity teams in alignment with their current operations.

References

[1] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, "Technical Guide to Information Security Testing and Assessment," National Institute of Standards and Technology, Tech. Rep. NIST SP 800-115, 2008.

[2] P. Vats, M. Mandot, and A. Gosain, "A comprehensive literature review of penetration testing & its applications," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2020, pp. 674–680.

[3] National Institute of Standards and Technology, "The NIST cybersecurity framework (CSF) 2.0," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST CSWP 29, Feb. 2024.

[4] European Union Agency for Cybersecurity (ENISA), "ENISA threat landscape 2025," European Union Agency for Cybersecurity (ENISA), Tech. Rep., Oct. 2025, doi: 10.2824/1946374.

[5] F. Di Franco and A. Grammatopoulos, "European cybersecurity skills framework (ECSF) user manual," European Union Agency for Cybersecurity (ENISA), Tech. Rep., Sep. 2022, doi: 10.2824/95989.

[6] European Union Agency for Cybersecurity (ENISA), "European cybersecurity skills framework role profiles," ENISA, Tech. Rep., Sep. 2022, doi: 10.2824/859537.

[7] A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, E. Zhu, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber, P. Chang, R. Loynd, R. West, V. Dibia, A. Awadallah, E. Kamar, R. Hosn, and S. Amershi, "Magentic-one: A generalist multi-agent system for solving complex tasks," 2024, doi: 10.48550/arXiv.2411.04468.

[8] T. Masterman, S. Besen, M. Sawtell, and A. Chao, "The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey," 2024, doi: 10.48550/arXiv.2404.11584. [Online]. Available: https://arxiv.org/abs/2404.11584

[9] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *Proc. 33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA, USA: USENIX Association, 2024, pp. 847–864.

[10] A. Happe, A. Kaplan, and J. Cito, "LLMs as hackers: Autonomous linux privilege escalation attacks," 2023.

[11] R. Fang, R. Bindu, A. Gupta, and D. Kang, "LLM agents can autonomously exploit one-day vulnerabilities," 2024.

[12] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "PentestAgent: Incorporating LLM agents to automated penetration testing," *arXiv preprint arXiv:2411.05185*, 2024.

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

[14] Alias Robotics, "CAI: Cybersecurity AI framework," https://github.com/aliasrobotics/cai, 2024, cai-framework v0.5.10.

[15] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, Q. Du, S. Shi, H. Mao, X. Zeng, and R. Zhao, "TPTU: Task planning and tool usage of large language model-based AI agents," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[16] LangChain AI, "LangGraph: Agent engineering platform," https://github.com/langchain-ai/langgraph, 2024.

[17] Rapid7, "Metasploitable 2: Intentionally vulnerable virtual machine for security testing," https://docs.rapid7.com/metasploit/metasploitable-2/, 2012, accessed: Jan. 21, 2026.

[18] Qwen Team, "Qwen3 technical report: Advancing open models to the next frontier," Alibaba Group, 2025.

[19] Google DeepMind, "Gemma 4 model card," https://ai.google.dev/gemma/docs/releases, 2026.

[20] OpenAI, "Introducing GPT-5," https://openai.com/index/introducing-gpt-5/, 2025.

[21] Ollama Team, "Ollama," https://github.com/ollama/ollama, 2026.