Cyber-Agent-Flow: Execution Trace Instrumentation and Analysis for Cybersecurity Agent Workflows (Extended Abstract)

Jaime C Acosta[1], Mohammad Taneem Nazim[2], Thomas Guerra[2], Yinuo Du[2], Palvi Aggarwal[2]

[1]Army Research Laboratory
[2]University of Texas at El Paso
jaime.c.acosta.civ@army.mil, mtnazim@miners.utep.edu, tnguerra@miners.utep.edu, ydu@utep.edu, paggarwal@utep.edu

Abstract

Large language models (LLMs) are increasingly explored as agents capable of assisting with complex cybersecurity tasks such as reconnaissance, vulnerability discovery, and penetration testing. Recent systems demonstrate that LLMs can coordinate multi-step workflows by combining reasoning with tool execution. However, most current evaluations focus primarily on task completion and provide limited insight into how agents behave during real-world engagements.

We present an instrumentation framework, *Cyber-Agent-Flow*, designed to improve the observability and analysis of agent-driven workflows. We implement this framework in the cybersecurity domain by recording detailed execution traces, including prompts, model responses, tool invocations, command outputs, and analyst annotations from penetration testing engagements. The framework integrates local cybersecurity tools through Model Context Protocol (MCP) interfaces while supporting privately hosted language models. Applying this framework to multi-step pivoting scenarios, we collect execution traces and analyze agent reasoning and tool orchestration. The framework reduces time and effort by consolidating information collected across the network and automating the penetration testing workflow. Additionally, observations reveal inefficiencies including redundant reasoning loops, repeated tool invocations, and suboptimal sequencing strategies — highlighting opportunities for improving the reliability and efficiency of agent-assisted cybersecurity operations.

Introduction

Large language models (LLMs) have enabled a new generation of agentic AI systems capable of reasoning about tasks, interacting with external tools, and executing multi-step workflows. In cybersecurity, these agents are increasingly explored for tasks such as penetration testing, vulnerability analysis, and incident response. Early systems such as PentestGPT demonstrate that LLMs can coordinate complex offensive workflows by iteratively generating commands, interpreting results, and adapting strategies during an attack process (Deng et al. 2024).

Despite these advances, evaluating such systems in realistic cyber environments remains a challenge. Most prior work measures performance using outcome-based metrics such as solved challenges or successful exploits, providing limited insight into how agents behave during execution or how efficiently they operate in practice.

LLM-driven cyber agents typically rely on repeated prompt–response cycles combined with external tool execution. Inefficiencies within this interaction loop—such as redundant reasoning cycles, overly long prompts, or poorly sequenced tool invocations—can significantly affect performance. Without visibility into these behaviors, diagnosing limitations and improving agent workflows becomes difficult, particularly in operational environments where transparency and human oversight are required.

In addition, many existing systems rely on cloud-hosted LLMs accessed through public APIs, introducing potential risks of exposing sensitive prompts and internal network information, which motivates the use of fully local, controlled deployments.

This work focuses on improving the observability and evaluation of LLM-driven cybersecurity agents. We introduce a framework, *Cyber-Agent-Flow*, for collecting detailed execution traces during agent runs, including prompts, model responses, tool invocations, artifacts, and timing information. The contributions of this work are:

- A framework for instrumenting LLM-driven cybersecurity agents and collecting detailed prompt and execution traces.
- A dataset capturing prompts, responses, tool executions, artifacts, and timing information from agent-driven penetration testing workflows.
- Preliminary observations identifying inefficiencies in prompt construction, reasoning loops, and tool orchestration.

Background and Related Work

Recent work has explored using Large Language Model (LLM) agents to automate penetration testing workflows. PentestGPT (Deng et al. 2024) demonstrated that LLMs can coordinate multi-step offensive tasks such as reconnaissance and exploitation through iterative prompt-driven tool interaction, while PentestAgent (Shen et al. 2024) integrates retrieval-augmented generation and multi-agent coordination for vulnerability analysis. IAPTF (Ghanem and Chen 2023) investigates automated penetration testing using reinforcement learning to navigate attack paths in large net-

works. However, analyses of LLM penetration testing agents highlight persistent challenges, including inefficient planning and context management limitations (Deng et al. 2026).

A related line of work evaluates LLM agents in Capture-the-Flag (CTF) environments. Systems such as EnIGMA, HackSynth, and CTFAgent demonstrate that LLM agents can solve cybersecurity challenges when combined with planning modules and specialized tools (Aletkin et al. 2024; Muzsai, Imolai, and Lukács 2024; Ji et al. 2025). More recent benchmarks examine realistic vulnerability exploitation scenarios, such as CVE-Bench (Zhu et al. 2025), evaluating whether agents can exploit real-world vulnerabilities under controlled conditions. However, most evaluations emphasize success metrics (e.g., solved challenges or exploit completion), providing limited insight into the reasoning and tool interactions that produce these outcomes.

More recently, research has explored tighter integration between LLM agents and penetration testing tools. Frameworks such as HexStrike (HexStrike AI 2026) investigate automated offensive workflows driven by LLM reasoning and MCP-based tool access. The Model Context Protocol (MCP) provides a standardized interface for connecting LLM applications to external tools (Anthropic and MCP Contributors 2025), and environments such as Kali Linux expose penetration testing utilities through MCP servers (Offensive Security 2026). While these efforts advance automated workflows, many rely on cloud-hosted LLM services and provide limited visibility into agent behavior during execution. Our work addresses both limitations through local LLM deployment and structured execution trace collection.

Cyber-Agent-Flow: Framework Overview

We developed a framework for studying and instrumenting LLM-driven cybersecurity agents operating in realistic penetration testing environments. The framework, referred to as *Cyber-Agent-Flow*[1], supports penetration testing workflows while providing visibility into how agents reason, interact with security tools, and progress through multi-step engagements as illustrated in Figure 1.

Human Input: The platform is accessed through a web-based interface, WebUI Frontend, that supports human-in-the-loop penetration testing workflows. This interface is served through a Flask-based orchestration layer, making the system accessible through the WebUI rather than a standalone CLI. It enables analysts to issue natural-language objectives, observe agent actions in real time, and provide annotations throughout execution. To support longer engagements, the system manages context by enforcing a configurable context budget, summarizing older interaction history while preserving the most recent exchanges in their original form. Furthermore, analysts can predetermine the tools that the agent can access during its pentest. It also supports analyst-defined constraints on tool availability and network access through selectable tool permissions and IP allowlists or blocklists, helping ensure controlled and policy-compliant execution. This helps the agent to provide better

[1]Available at https://raistlinj.github.io/cyber-agent-flow-site/.

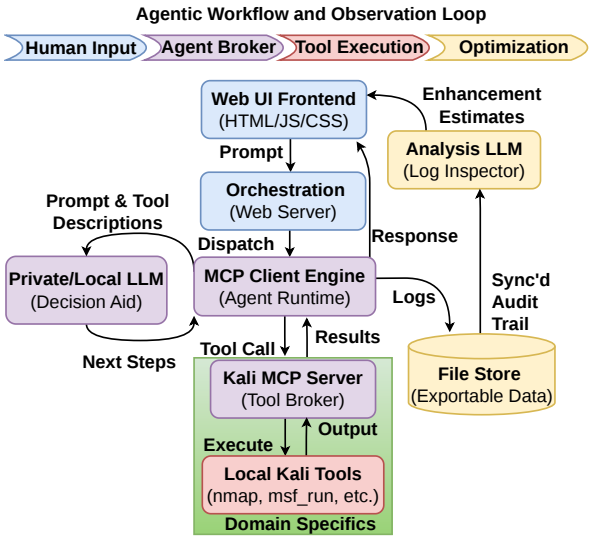

Figure 1: Architecture of the Cyber-Agent-Flow platform supporting human-in-the-loop orchestration and telemetry collection.

insights into potential improvements during and after a session.

Agent Broker: The Agent Broker functions as the system's central control component with several coordinated responsibilities. The MCP Client Engine operates beyond forwarding prompts to the LLM model, it also invokes the corresponding MCP servers and necessary tools to execute actions derived from model responses. Additionally, it logs data from its actions and stores it for use by the Analysis LLM. Finally, it provides a final response to the WebUI consisting of the action and feedback on the environment with suggestions for optimization. Many recent LLM-driven penetration testing systems rely on cloud-hosted models accessed through public APIs, which can expose sensitive vulnerability data and network telemetry. In contrast, our architecture operates entirely within a self-contained environment using a local language model via the Ollama runtime (Ollama 2023), ensuring privacy-preserving operation. The model interacts with penetration testing tools exposed through Model Context Protocol (MCP) interfaces, while a LiteLLM (BerriAI 2023) gateway provides request routing and access control.

Tool Execution: This component takes the instructions from the agent broker to run the local Kali tools available in the system. The outputs generated from running the command are sent back through the MCP server to the agent for further reasoning and its corresponding steps.

Optimization: A key component of the system is an integrated observation and telemetry layer that records the full interaction lifecycle of each session. Prompts, model responses, tool invocations, command outputs, analyst annotations, and timing information are captured as structured ex-

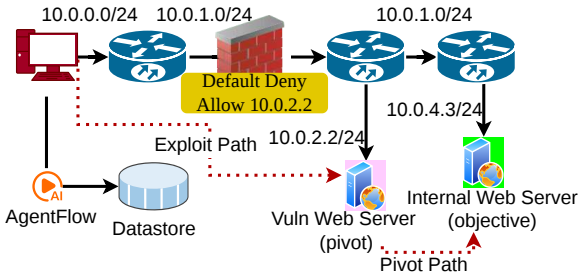

Figure 2: Pivoting scenario used to generate execution traces.

ecution traces. These traces form the dataset which the Optimizer engine (backed by an analysis LLM) inspects agent behavior and identifies workflow inefficiencies. All sessions are archived as structured datasets containing the full interaction lifecycle of an engagement.

Exploratory Study

Experimental Setup

To demonstrate the framework's collected data and trace analysis capabilities, we conducted a multi-step penetration testing engagement using a pivoting scenario (Acosta et al. 2021). In this scenario, an attacker uses a compromised host to reach otherwise inaccessible internal systems through sequential reconnaissance, lateral movement, and tool chaining.

As shown in Figure 2, the agent starts from an externally reachable host, performs reconnaissance, and pivots through intermediary systems to reach protected internal services. These engagements were executed using the Cyber-Agent-Flow platform, enabling the system to record detailed telemetry of the agent's reasoning process, tool interactions, and analyst annotations.[2] We then used the analysis engine to identify potential efficiencies for similar scenarios.

Results

The agent successfully completed the scenario defined in Figure 2. As part of its post-hoc analysis functionality, Cyber-Agent-Flow can analyze archived execution traces and generate summaries or performance estimates based on the recorded telemetry. To quantify the agent's performance, we used this functionality to estimate time spent and number of interactions across the primary stages of the attack workflow. Using the instrumented telemetry logs as input, the model estimated the time spent in each phase of the interaction, including reconnaissance, service identification, exploit selection, exploit execution, and pivot configuration (Figure 3). The results indicate substantial variation in completion time across workflow stages, with the largest share observed in exploit selection, exploit attempts, and pivot setup. The collected traces further suggest that these inefficiencies stem from redundant tool calls, repeated exploit

[2]Dataset is available for download at https://github.com/raistlinJ/cyber-agent-flow/blob/main/datasets/pivot-sample.zip.

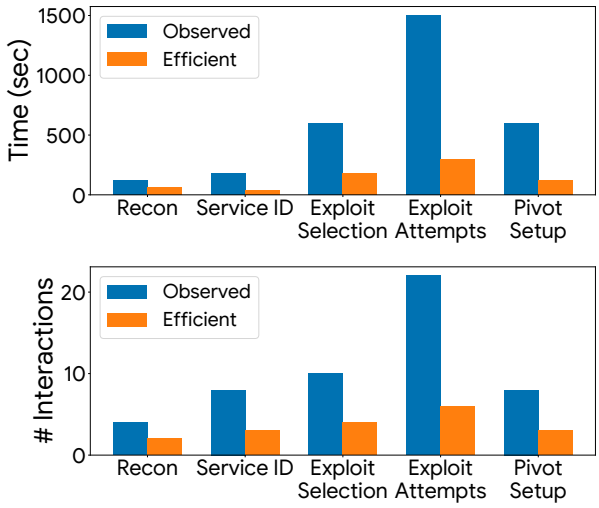

Figure 3: Estimated agent performance across interaction phases. Top: task completion time. Bottom: number of interactions (tool invocations and reasoning steps).

attempts, and clarification loops, as evidenced by the high number of interactions.

Projected Time Savings (Efficient Agent). In order to assess how Cyber-Agent-Flow informs workflow optimization, we produced a counterfactual estimate by prompting the LLM to project execution time under conditions where the identified failure patterns are absent. We define *estimated efficient execution* as a counterfactual scenario in which hallucinations are minimized, tool usage is accurate, and lightweight automation reduces interaction overhead (Figure 3).

Furthermore, we prompted the LLM to estimate a hypothetical execution time assuming the elimination of redundant tool invocations, repeated exploit attempts, and command–context errors.

As shown in Figure 3, the LLM estimated that the observed workflow required approximately 50 minutes for the agent to reach the pivot stage of the attack. In contrast, an optimized workflow that eliminates redundant scans, limits exploit retries, and correctly manages tool context is estimated to require approximately 11 minutes. This observation highlights an important challenge for agentic cybersecurity systems: effective orchestration of existing tools may be as critical as the reasoning capabilities used to select them. To better understand the agent's hallucinations, we report the observed failure patterns in the following section.

Failure Patterns

Inspection of the execution traces reveals several recurring patterns that negatively affect workflow efficiency. The following observations are derived from a single run and should not be interpreted as statistically representative, but illustrate common challenges encountered during agent-driven penetration testing tasks.

Redundant tool invocations. The agent occasionally executes identical reconnaissance commands multiple times (e.g., repeating network scans with unchanged parameters) even when previous results remain valid. These redundant operations increase execution time without providing additional information.

Command-context confusion. Penetration testing tasks involve multiple execution environments, including the system shell, Metasploit console, and Meterpreter sessions. The agent sometimes generates commands intended for one environment while operating in another, including hallucinated commands or incorrect suggestions caused by tool version differences. This often leads to execution failures and additional reasoning steps to correct the mismatch.

Exploit retry loops. When exploitation attempts fail or time out, the agent may sequentially attempt multiple exploit modules with minimal parameter adjustments. In several cases, similar exploit configurations are retried without incorporating new information from prior failures.

Environment constraint violations. Some generated commands assume capabilities that are unavailable in the current environment (e.g., unsupported shell operators or restricted commands), resulting in execution errors that require correction.

Pivot configuration complexity. Establishing network pivot routes requires coordination between Meterpreter session commands and Metasploit modules. The trace suggests that agents may struggle to distinguish between routing commands available within each environment, resulting in additional trial-and-error during pivot setup.

Conclusion

Large language model agents have emerged as promising tools in assisting with complex cybersecurity workflows. However, their effectiveness depends not only on whether they complete a task, but also on how they reason, interact with tools, adapt to failures, and operate across multi-step engagements.

In this work, we introduced Cyber-Agent-Flow, a framework for capturing structured execution traces that improve visibility into agent behavior during penetration testing workflows. While PentestGPT provides an important foundation through its reasoning architecture and explicit task-state representation, it primarily operates as an interactive guidance framework in which a human executes actions and returns results to the model. In contrast, our design embeds the agent within a local executable environment, enabling bounded autonomous tool interaction, timeout handling, and iterative command adaptation. The system collects detailed execution traces capturing prompts, model responses, tool invocations, artifacts, and analyst annotations, shifting the focus from human-mediated feedback to direct observation of workflow behavior.

These findings highlight the importance of detailed telemetry for studying LLM-driven cybersecurity agents. By providing visibility into how agents reason and interact with tools during complex engagements, the framework enables more systematic analysis and improvement of agent-assisted penetration testing workflows. A preliminary analysis identified inefficiencies related to reasoning loops, tool orchestration, and execution context management in a multi-step pivoting scenario.

Future work will expand the dataset to include additional penetration testing scenarios and more complex network environments, enabling more systematic analysis across attack stages and infrastructure configurations. We also plan to enhance analysis through automated and human-in-the-loop techniques, including detection of recurring inefficiencies (e.g., redundant tool use and ineffective reasoning loops) and comparison of agent workflows with established penetration testing practices to improve efficiency and realism.

References

Acosta, J. C.; Medina, S.; Ellis, J.; Clarke, L.; Rivas, V.; and Newcomb, A. 2021. Network data curation toolkit: cybersecurity data collection, aided-labeling, and rule generation. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, 849–854. IEEE.

Aletkin, A.; et al. 2024. EnIGMA: Interactive Tools Substantially Assist LM Agents in Finding Security Vulnerabilities. arXiv:2409.16165.

Anthropic and MCP Contributors. 2025. Model Context Protocol Specification. Online specification.

BerriAI. 2023. LiteLLM.

Deng, G.; et al. 2024. PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. In *Proceedings of the USENIX Security Symposium*.

Deng, G.; et al. 2026. What Makes a Good LLM Agent for Real-world Penetration Testing? arXiv:2602.17622.

Ghanem, M. C.; and Chen, T. M. 2023. Hierarchical Reinforcement Learning for Efficient and Effective Automated Penetration Testing of Large Networks. *Journal of Intelligent Information Systems*.

HexStrike AI. 2026. HexStrike AI MCP Agents v6.0. GitHub repository.

Ji, X.; et al. 2025. Measuring and Augmenting Large Language Models for Solving Capture-the-Flag Challenges. arXiv:2506.17644.

Muzsai, L.; Imolai, D.; and Lukács, A. 2024. Hacksynth: LLM agent and evaluation framework for autonomous penetration testing. *arXiv preprint arXiv:2412.01778*.

Offensive Security. 2026. mcp-kali-server — Kali Linux Tools. Kali Linux tools documentation.

Ollama. 2023. Ollama.

Shen, X.; et al. 2024. PentestAgent: Incorporating LLM Agents to Automated Penetration Testing. arXiv:2411.05185.

Zhu, Y.; et al. 2025. CVE-Bench: A Benchmark for AI Agents' Ability to Exploit Real-World Web Application Vulnerabilities. arXiv:2503.17332.