ARL-TR-10488 ● Nov 2026



DEVCOM ARMY RESEARCH LABORATORY

Cyber-Agent-Flow: Enhancing Cybersecurity Effectiveness through Human–AI Teaming for Cybersecurity Workflow Optimization

by Thomas Guerra and Jaime C. Acosta

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-10488 ● Nov 2026





Cyber-Agent-Flow: Enhancing Cybersecurity Effectiveness through Human–AI Teaming for Cybersecurity Workflow Optimization

Thomas Guerra
University of Texas at El Paso

Jaime C. Acosta
DEVCOM Army Research Laboratory

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
November 2026	Technical Report	June 2026	August 2026

4. TITLE AND SUBTITLE
Cyber-Agent-Flow: Enhancing Cybersecurity Effectiveness through Human–AI Teaming for Cybersecurity Workflow Optimization

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER ARL CRADA 14-0011-016-A01	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Thomas Guerra and Jaime C. Acosta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-ND Adelphi, MD 20783-1138	ARL-TR-10488

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID: Jaime C. Acosta, 0000-0003-2555-9989

14. ABSTRACT
Large language models (LLMs) are increasingly used as agents for assisting with cybersecurity testing. While recent systems show that LLMs can coordinate multistep workflows combining reasoning and tool use, most evaluations focus only on task completion and provide limited insight into why agents succeed or fail in realistic environments.
We present Cyber-Agent-Flow, an instrumentation and iterative workflow-optimization framework for autonomous cybersecurity agents. It records detailed execution traces, including prompts, model responses, Model Context Protocol (MCP) tool invocations, command outputs, and analyst annotations, while integrating local tools and supporting privately hosted models. To enable controlled evaluation, we apply this framework within controlled enterprise network emulations built using the Common Open Research Emulator and Proxmox with six progressively complex enterprise scenarios, from single-host validation to multistep lateral movement. Deterministic deployments isolate agent behavior as the sole experimental variable.

15. SUBJECT TERMS
cybersecurity, scan tools, intrusion detection, Collaborative Innovation Testbed, CyberRIG, Network, Cyber and Computational Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	28

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Jaime C. Acosta	(520) 691-5956

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures v

List of Tables v

1. Introduction 1

1.1 Related Work 1

1.2 Prior Work 2

1.3 Contributions of This Report 2

2. Cyber Range and Evaluation Environment 3

2.1 Four-Layer Scenario Architecture 4

2.2 Scenario Progression Philosophy 4

2.3 Scenario Overview 5

3. Cyber-Agent-Flow: Instrumentation Platform 9

3.1 Architecture 9

3.2 Engagement Workflow 10

4. Recommendations Pipeline: Workflow Optimization Loop Contribution 10

4.1 From Static Findings to a Working Interface 11

4.2 Persistent Storage for Jobs and Analyst Notes 11

4.3 Achieving Provider-Agnostic Generation 11

4.4 Run Analysis 12

4.5 Plugins System for Generated Tools and Playbooks 12

4.6 End-to-End Workflow 13

5. Experimental Setup and Preliminary Findings 13

5.1 Library 14

5.2 Preliminary Agent Evaluation 15

5.3 Preliminary Findings 15

6. Future Work **16**

7. Conclusion **18**

8. References **19**

List of Symbols, Abbreviations, and Acronyms **20**

Distribution List **21**

List of Figures

Figure 1. Attacker network configuration. 6
Figure 2. Single test. 6
Figure 3. Home network with IoT devices. 7
Figure 4. Small-business flat network. 7
Figure 5. Segmented small business. 8
Figure 6. Hospital segmented enterprise network. 9

List of Tables

Table 1. Cyber scenarios. 5
Table 2. Public vuln data 14

1. Introduction

Public reporting and research suggest that threat actors are increasingly leveraging AI to enhance cyber operations, including the Ransomware-as-a-Service ecosystems and AI-generated malware. As a result, lower-skilled adversaries are now capable of launching highly sophisticated and scalable attack campaigns relatively quickly, even against targets that were previously considered too "low value" to prioritize.

This research addresses this issue directly with our own large language model (LLM)-powered cybersecurity framework to assist with the following:

- **Intelligent Tool Chaining:** Automates repetitive security tasks while capturing structured logs of tool execution and results.
- **Local LLM Execution:** Ensures data sovereignty/privacy by operating entirely within a closed, local environment.
- **Human-in-the-Loop Control:** Supports analyst decision-making through AI-assisted reasoning and guided workflow execution.

1.1 Related Work

Recent advances in LLMs have demonstrated significant offensive potential. One instance recently was Claude ranking in the top 3% of the 2025 PicoCTF, outperforming many human teams in speed (Anthropic 2025). These results highlight that LLMs can effectively automate portions of testing.

Similarly, other work introduced LLMs as "sparring partners" capable of high-level planning and autonomous testing through closed feedback loops (Happe and Cito 2023). This research evolved into frameworks like HackingBuddyGPT, where GPT-3.5-Turbo succeeded with 16%–50% of tests, significantly outperforming local models (Happe et al. 2026). Due to the initial success and promise of AI penetration testing agents, the field has since matured into operational, dockerized pipelines such as Strix, PentestGPT, and CAI (Mader 2026). However, despite these gains, autonomous agents still struggle with inefficient reasoning and repetitive actions during multistage attacks (Singer et al. 2025). The Incalmo framework addresses these core limitations by decoupling strategic planning from tactical execution. Although it represents an evolving approach rather than a complete remedy, its use of structured abstraction layers allows the system to maintain a high-level understanding of the objective throughout complex tasks (Singer et al. 2025).

While these studies confirm LLM capabilities, evaluations typically focus on task outcomes rather than execution efficiency (Anthropic 2025; Singer et al. 2025; Happe et al. 2026). Inefficient workflow-level behaviors, like suboptimal tool sequencing and repeated commands, remain critical areas for investigation in cyber research with AI (Singer et al. 2025; Happe et al. 2026). This key gap motivates the development of Cyber-Agent-Flow, which focuses on instrumenting and analyzing the execution processes of AI-driven cybersecurity operations to identify systematic opportunities for improvement.

1.2 Prior Work

The Cyber-Agent-Flow framework shifts the focus from simple outcome-based metrics to deep observability by providing a dedicated instrumentation platform for autonomous cybersecurity workflows. Its architecture pairs a WebUI Frontend for human-in-the-loop orchestration with a Model Context Protocol (MCP) Client Engine, enabling local, privacy-preserving LLMs to interact with security tools via the MCP. During a multistep cyber range pivoting scenario, the platform captured a "Sync'd Audit Trail," including prompts, tool outputs, and timing data, as an agent performed reconnaissance, lateral movement, and pivot configuration to reach a protected internal service (Acosta et al. 2026).

Analysis of these execution traces, drawn from a single exploratory run, revealed that while the agent could complete the task, it exhibited operational inefficiencies including redundant tool calls, command-context confusion (such as mixing up the system shell with the Metasploit console), and repeated retry loops. Notably, the study estimated that the agent required approximately 50 min to reach the pivot stage, whereas a projected, optimized workflow eliminating these failure patterns could theoretically complete the same task in roughly 11 min (Acosta et al. 2026). Ultimately, Cyber-Agent-Flow demonstrates that refining agentic orchestration and tool sequencing is as critical to success as a model's core reasoning capabilities, though these findings are illustrative rather than statistically representative. Even so, the results demonstrate clear promise for the kind of systematic workflow optimization this report extends.

1.3 Contributions of This Report

As discussed in the previous section, Cyber-Agent-Flow was established as an instrumentation platform capable of capturing detailed execution traces from LLM-driven penetration testing engagements and surfacing workflow inefficiencies through post-operation analysis. That analysis, however, produced findings that existed only as static text that generated recommendations but had no path from

observation to action. This report documents the work done to extend Cyber-Agent-Flow by closing that post-operation optimizations gap. Building on the existing analysis pipeline, this work introduces a complete Recommendations and Plugins system that turns post-operation findings into reusable capabilities for future engagements. Specifically, this report's contributions are as follows:

- A Recommendations interface that surfaces every analyzed engagement's findings as reviewable cards, giving analysts a dedicated space to inspect identified inefficiencies and candidate improvements rather than leaving them buried in analysis output.
- A human-in-the-loop generation pipeline that allows an analyst to review, annotate, and approve a recommendation before it is turned into a new MCP tool or Markdown playbook, which replaced an earlier command-line interface (CLI)-dependent generation process with a "provider-agnostic" approach that works consistently across locally hosted and cloud-hosted models.
- A Plugins system for tracking, storing, and safely reusing AI-generated tools and playbooks, including collision protection, deterministic provenance recording, and job tracking that persists across restarts, so every generated capability remains traceable to the engagement and analyst decision that produced it.
- Integration into live sessions, allowing previously generated tools and playbooks to be selected and used in future engagements, closing the loop between what one engagement reveals and what the next engagement can act on.

Together, these contributions shift Cyber-Agent-Flow from a system that observes and reports on agent behavior into one that supports iterative, analyst-supervised improvement of that behavior over time, all without altering the underlying language model itself.

2. Cyber Range and Evaluation Environment

To evaluate autonomous cybersecurity agents under controlled conditions, this research uses the Common Open Research Emulator (CORE 2026), a platform that instantiates realistic topologies using Linux kernel namespaces. Cyber-Agent-Flow deploys against these environments to safely and ethically conduct experiments and harvest execution data. Unlike a network simulator, which models behavior mathematically, an emulator like CORE runs functional operating systems, services, and routing daemons. This distinction is critical because each node runs unmodified software, so agents must navigate genuine weak services, firewall

rules, and routing behaviors. An agent within a CORE topology reasons about the same conditions it would encounter on a physical network rather than a mathematical approximation.

2.1 Four-Layer Scenario Architecture

Each scenario in this cyber range is built from four independent components, allowing network design to be decoupled from security posture. The four components are as follows:

- **Network Topology:** the routers, switches, links, addressing, and routing that define which nodes exist and how they communicate.
- **Node Provisioning:** the operating systems, users, credentials, and services present on each node.
- **Security Configuration:** the firewall rules, network address translation (NAT) behavior, and access control policies that define the boundaries an attacker must navigate.
- **Startup Automation:** scripts that deploy every component identically on every run, ensuring deterministic, reproducible environments without manual reconfiguration between evaluations.

This separation of independent components is what allows the same physical topology to represent either a correctly secured environment or a deliberately misconfigured one by changing only the security configuration layer.

2.2 Scenario Progression Philosophy

Scenarios in this cyber range (Table 1) are not ordered by increasing difficulty alone. Each is designed to evaluate a distinct networking or security concept, so that an agent's performance can be traced to a specific capability rather than a general notion of "harder."

Table 1. Cyber scenarios.

Scenario	Research goal	Security concept
1	Enumerate a firewall and test an exposed service	NAT, port forwarding, service discovery
2	Reason about a home network with Internet of Things (IoT) devices	Flat network, device co-location, lateral reach
3	Flat enterprise discovery testing	No segmentation, full lateral movement
4	Enterprise segmentation and access control	Defense-in-depth, inter-subnet policy
5	Segmentation failure and lateral movement	Misconfiguration vs. correct configuration
6	Healthcare enterprise architecture	Multizone segmentation, demilitarized zone (DMZ), strict access control

Environments remain fixed once established and run on the Debian12 device hosting CORE. The only change during experimentation and data collection would ideally be the AI framework under evaluation between runs, through improved prompts, new MCP tools, or workflow optimizations generated. This ensures that any measured improvement in agent performance can be attributed to the framework itself rather than to environmental variation.

2.3 Scenario Overview

Across all six scenarios, the attacker-facing side of the topology remains identical, isolating the victim-side network as the only variable under test. Each scenario connects the emulated attacker nodes, reachable from the Kali Linux attacker platform, through a dedicated attacker switch and attacker router (Figure 1). This router is the fixed entry point into every scenario's victim environment emulating a network hop via the web from the attacker to victim networks.

- This attacker-side segment is designed to simulate a remote adversary rather than an attacker with direct local access. The attacker node reaches the target environment through an emulated routing hop, mirroring how a real internet-based attacker would approach a target network from outside its perimeter rather than from within it.

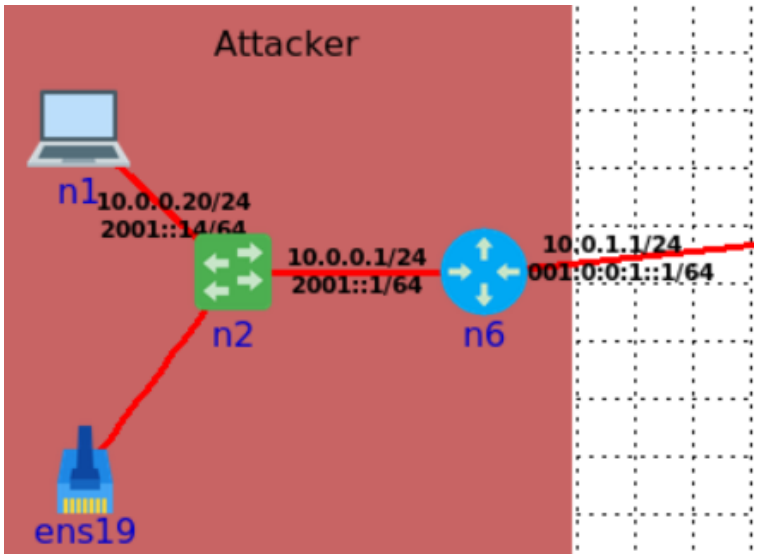


Figure 1. Attacker network configuration.

- **Scenario 1** is a home-user network with a single internet-connected device running one service, exposed through NAT-based port forwarding (Figure 2). This scenario is intentionally not difficult, and exists to establish a known good baseline, validating routing, firewall behavior, and execution before complexity increases in later scenarios.

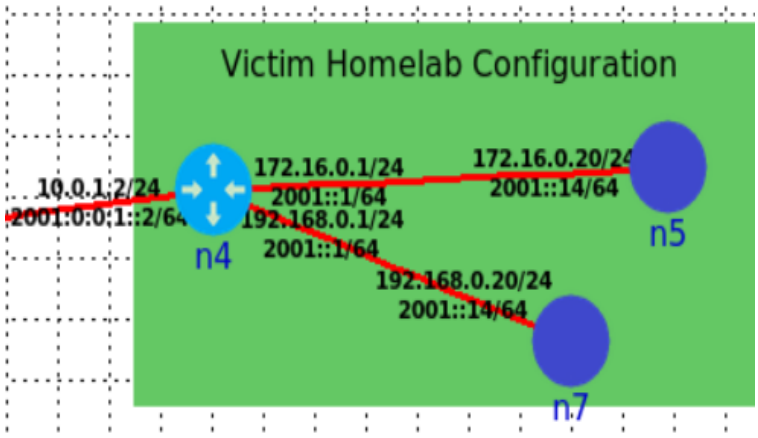


Figure 2. Single test.

- **Scenario 2** is a residential network with a clean workstation and two IoT-style nodes sharing a single flat subnet, introducing lateral movement where compromising one IoT node grants direct reach to the workstation (Figure 3).

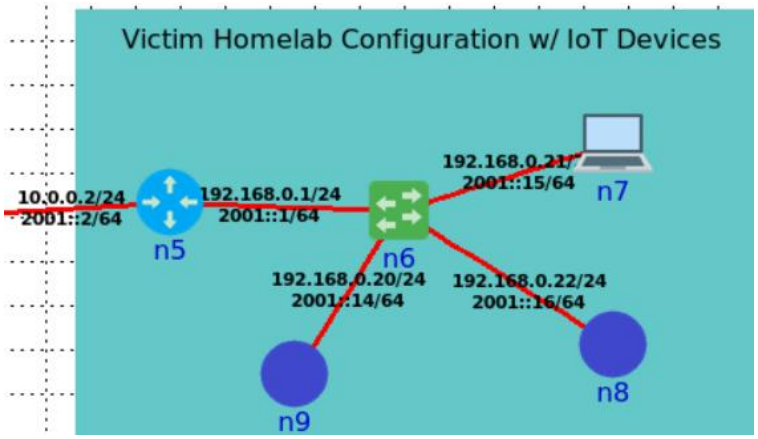


Figure 3. Home network with IoT devices.

- **Scenario 3** is a small-business network with all infrastructure, including but not limited to workstations, servers, backup systems, and IoT devices, all on a single unsegmented subnet, where compromising any one device grants unrestricted access to every other asset (Figure 4).

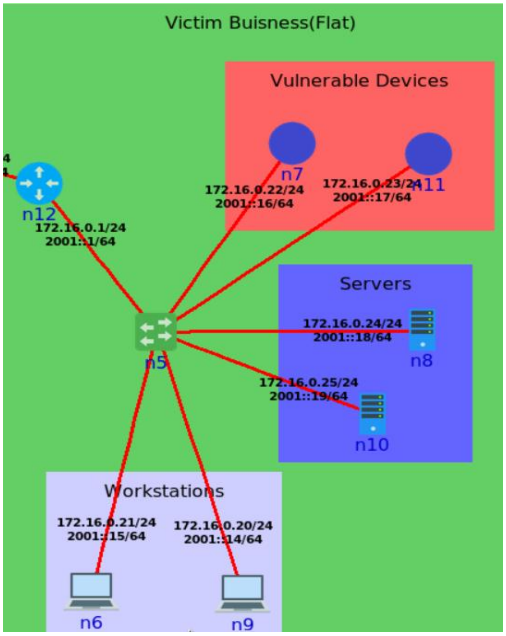


Figure 4. Small-business flat network.

- **Scenario 4** has the same asset composition as Scenario 3 but is restructured across three dedicated subnets connected through a central router enforcing inter-subnet access control, so that compromising an IoT device contains the attacker within that segment.
- **Scenario 5** (misconfigured) is physically identical to Scenario 4, but with inter-subnet access controls removed and node-level security intentionally weakened. This demonstrates that segmentation infrastructure (Figure 5)

without enforced policy produces the same effective attack surface as a flat network, and tests whether the agent can recognize and test that weakness efficiently rather than defaulting to brute-force enumeration.

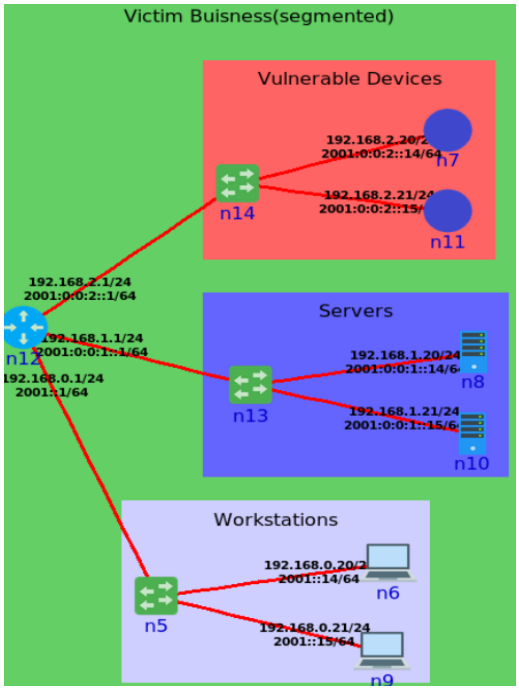


Figure 5. Segmented small business.

- **Scenario 6** is a multizone healthcare network with clinical, administrative, imaging, medical IoT, and server segments behind a perimeter firewall and DMZ, using dual-stack IPv4/IPv6 addressing (Figure 6). Rather than treating the firewall purely as an obstacle, this scenario treats it as an observable component of the environment. An important part of what the agent must reason about is recognizing filtering behavior, identifying which services are intentionally exposed and adapting reconnaissance accordingly, mirroring how a real enterprise network presents itself to an external attacker.

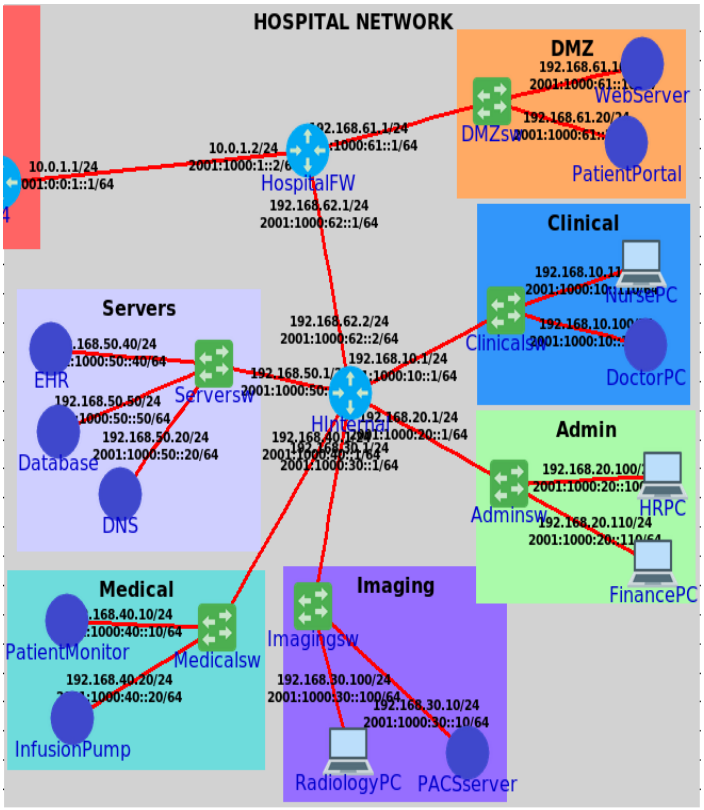


Figure 6. Hospital segmented enterprise network.

3. Cyber-Agent-Flow: Instrumentation Platform

Cyber-Agent-Flow is a privacy-preserving, fully localized agentic penetration testing platform that bridges LLMs and native cybersecurity toolchains through MCP. Rather than functioning as a simple command-execution wrapper, the platform operates as a cybersecurity workflow engine. This system collects detailed telemetry across live engagements (LLM reasoning trajectories, raw tool outputs, network captures, and analyst annotations) enabling both in-session tactical guidance and post-engagement methodology review. All orchestration runs through a locally hosted LLM instance, ensuring that proprietary network layouts, data, and engagement findings never leave the operator's environment.

3.1 Architecture

The platform is organized into four key layers—a browser-based WebUI that provides live chat, session browsing, and analysis tools, communicating with a Flask middleware layer over REST and Server-Sent Events. This middleware routes requests to the MCP Client Engine, which manages conversation context, issues calls to the LLM backend, and dispatches tool invocations. Tool execution itself is handled by a dedicated MCP server exposing native Kali Linux utilities,

like network scanners, testing frameworks, packet capture tools, and related utilities, as callable functions the agent can invoke directly. The LLM backend runs through Ollama or an Ollama-compatible endpoint, keeping inference local by default.

Every session is recorded to a structured, disk-backed run directory, capturing a full JSON event log alongside a human-readable Markdown transcript. Analysts can insert timestamped annotations mid-engagement to guide the agent or flag data for later review, and completed sessions remain browsable and exportable as self-contained archives.

3.2 Engagement Workflow

A typical evaluation using Cyber-Agent-Flow proceeds through four phases:

1) **Reconnaissance and Enumeration:** The agent identifies live hosts, enumerates exposed services, and collects the information required for attack planning by invoking native Kali tools through the MCP server and reasoning over their output.
2) **Test:** The agent executes "proof-of-vulns" against intentional system weaknesses, with every attempt, success, and failure recorded to the session's event log.
3) **Lateral Movement and Multistep Scenarios:** Where a scenario's topology permits, the agent leverages compromised credentials or system information to reach additional network resources, extending the engagement beyond a single host.
4) **Automated Analysis:** Cyber-Agent-Flow documents findings and analyzes the recorded attack path, surfacing tool usage patterns and reasoning behavior for review.

This last stage for post-engagement analysis can be initiated separately once a session has concluded. The analyst selects a completed engagement and triggers this automated analysis job, that will then feed the full session transcript back into an LLM to generate a narrative summary and identify methodological inefficiencies in the recorded workflow. This output is what the Recommendations pipeline described in Section 4 builds on.

4. Recommendations Pipeline: Workflow Optimization Loop Contribution

In the last section, we covered that Cyber-Agent-Flow could already review a completed engagement and produce text recommendations, including candidate

new tools, playbook ideas, and efficiency findings. That text had nowhere to go. It sat inside a job record with no dedicated place for an analyst to review it, decide whether to act on it, or turn it into something usable.

4.1 From Static Findings to a Working Interface

This work built a new Recommendations tab as a self-contained frontend module, following the same design pattern already established elsewhere in the platform rather than growing the main application file further. The tab surfaces every analyzed session as an expandable card. Expanding a card shows the full analysis text alongside each recommended asset, split into its own Markdown playbook and MCP tool sections. Several layout and styling issues had to be resolved along the way, including cards that had no dedicated styling at all when this work began, and a results list that was artificially capped to a small scrollable box originally designed for a different, more compact view. An early design attempt grouped cards by session rather than by individual analysis job. This was reverted after it incorrectly collapsed multiple distinct sessions together, and the final design returned to per-job cards with all placeholder text replaced by data parsed from the session's actual output.

4.2 Persistent Storage for Jobs and Analyst Notes

Two separate problems were found in how this data was being stored. Analysis jobs were only ever held in memory for the running application, so a server restart made the job list appear empty even though the underlying records were still saved to disk. On top of that, there was no real way for an analyst to leave commentary on a recommendation. An early notes' feature only saved to the browser's local storage, meaning a note was invisible to anyone else and could be lost entirely if that browser's storage was ever cleared.

Both issues were fixed by making disk the authoritative source of truth. The job list now scans the run directory directly rather than relying on memory, and analyst notes are written to a dedicated file per session, kept deliberately separate from anything AI generated. New endpoints allow notes to be added or removed, and those notes are merged into the job list automatically. Notes are also fed directly into future generation requests, so an analyst can correct or scope what the automated analysis got wrong before anything is built from it.

4.3 Achieving Provider-Agnostic Generation

The original mechanism for turning a recommendation into a working tool was built around launching an external command-line assistant as a separate interactive

process. This required a manual login step and a local proxy process for any model provider other than that one assistant, and a bug in how provider names were compared meant every provider, including the one the system was built around, incorrectly triggered that proxy path. On a fresh machine, this failed outright with missing dependencies and a login prompt that had no way to be answered in an automated context.

The generation path was rebuilt around the same provider-agnostic request method the analysis feature already used successfully. The new version sends a prompt, parses a simple file-block format out of the model's plain text response, and writes the resulting files to disk directly. There is no external process, no login step, and no dependency on which model provider is selected, as long as it is compatible with standard chat completion application programming interfaces (APIs) and the platform's MCP tool integration. The old interface for configuring that external assistant was replaced with a simpler generation form, prefilled with the recommendation's problem statement, expected benefit, and any analyst notes shown as a final read-only check before generation begins.

4.4 Run Analysis

Once generation no longer depended on an external process, a new problem appeared. There was no record of what had been generated, no protection against two different recommendations overwriting each other's output, and no way to know whether a given generated file could be trusted. To fix this, a dedicated storage location for generated tools and playbooks was built, kept separate from the platform's hand-curated tool catalog, since AI-generated code is a different trust category from tools a human has written and reviewed. If a generated asset's name already exists, the system refuses to overwrite it unless explicitly told to, protecting against two different engagements silently colliding on the same output. Every generated asset also receives a provenance record written by the platform itself, not self-reported by the model, recording when it was generated, which engagement it came from, and the full prompt used to produce it. Generation jobs are tracked the same way analysis jobs already are, persisted to disk so history survives a restart, and can be cancelled before any file is actually written to disk.

4.5 Plugins System for Generated Tools and Playbooks

A generated tool or playbook was still a dead end without a way to actually use it in a future session. A new configuration section was added, listing available generated tools and playbooks separately, populated directly from what exists on disk. Checking a generated tool merges its definition into the tool list sent when a

new session starts, and checking a playbook adds its guidance into the system prompt for that session. These selections persist across tab switches and page reloads, matching how the platform's existing tool selection already behaved.

4.6 End-to-End Workflow

Taken together, generated tools and playbooks close the loop the original platform was missing. An analyst runs a live engagement as described in Section 3. Once the session ends, an analysis job reviews the transcript and produces structured findings, including candidate new tools and playbooks. The analyst reviews those findings in the Recommendations tab, optionally adding notes to correct or scope what should happen next. Approving a recommendation generates the actual tool or playbook file, tracked as a background job and recorded with full provenance. That generated asset then appears in the platform's configuration options, available to be selected and used in a future engagement.

No step in this process happens automatically without a human decision in between. An engagement can now teach the system something specific to the inefficiencies it encountered, and a future engagement can use what was learned with every addition traceable back to the engagement and the analyst's judgment that produced it.

5. Experimental Setup and Preliminary Findings

All experiments were run on a Proxmox virtualization cluster hosting two dedicated virtual machines (VMs), each serving a distinct role in the evaluation pipeline. A Kali Linux VM runs Cyber-Agent-Flow itself, connected via a wired Ethernet connection into the simulated attacker network. A second VM, running Debian 12, hosts CORE, the Docker-based containers pulled from vulnhub, and the active scenario topology being tested.

The Kali attacker VM is strictly isolated at the network level. Its only permitted outbound path beyond the target CORE environment is a connection to a llama.cpp server hosted on the university network, running Qwen3.6-35B-A3B quantized to a 4-bit format (Q4_K_M) using the GPT-Generated Unified Format (GGUF) standard. Cyber-Agent-Flow connects to this server through its OpenAI-compatible provider setting, pointed at the server's address on the university network. Reaching this server required resolving its IP with domain information groper (dig) and adding an explicit route for it on the Kali VM, since the server is not reachable through default routing from the isolated attacker network.

This connection uses a dedicated interface completely separated from the one used to reach the emulated victim environment. This architectural separation is intentional, as it confines the agent's reasoning traffic to a controlled, single-purpose path while denying general internet access. This constraint ensures the agent cannot pull external proof-of-vulns from the web, forcing it to autonomously discover and reason. To support this infrastructure, the Debian 12 host running CORE connects to the same university segment over a dedicated physical interface, ensuring that the target scenario and the attacking agent remain bound to the same isolated lab network throughout the engagement.

5.1 Library

Seven test scenarios were confirmed and fully documented through the complete test chain, from the Kali attacker through the scenario router to the victim container, before being used in any agent evaluation (Table 2).

Table 2. Public vuln data.

Proof-of-vuln	CVE	Type
Apache Struts2 S2-045	CVE-2017-5638	Unauthenticated RCE via OGNL injection
Grafana path traversal	CVE-2021-43798	Unauthenticated file read
Ghostcat (Tomcat AJP)	CVE-2020-1938	Unauthenticated AJP file read
SambaCry	CVE-2017-7494	Authenticated RCE via shared object injection
Mongo Express	CVE-2019-10758	Default credentials plus Node.js injection
ownCloud info disclosure	CVE-2023-49103	Unauthenticated credential dump
Jupyter Notebook	Misconfiguration	Unauthenticated RCE via kernel API

Notes: CVE = common vulnerability and exposure; AJP = Apache JServe Protocol; RCE = remote code execution; OGNL = object-graph navigation language.

Each proof-of-vuln was documented independently, written to avoid line-for-line replication of publicly available write-ups, so that these entries can later serve as evaluation benchmarks where internet lookup or training-data recall alone would not be sufficient for an agent to succeed.

Not every candidate could be deployed successfully. Testing surfaced a recurring compatibility problem in CORE's container build layer. Several candidate containers were built on end-of-life Debian base images with expired GNU Privacy Guard (GPG) keys, causing the build step to fail outright before the container ever ran. This pattern is now understood well enough to be screened for before investing time in a new candidate, and known-compatible base images (Alpine, Ubuntu 20.04 and later, Debian Buster and later) are used for new candidates going forward.

5.2 Preliminary Agent Evaluation

Each scenario was evaluated against Cyber-Agent-Flow across a set of progressive assistance levels, from no guidance beyond authorized scope and target IP, up to being given the exact payload, which establishes an upper bound simulating what retrieval-augmented generation (RAG) could provide.

Against Struts2, the agent eventually achieved full RCE with root privileges, but only after significant inefficiency. It defaulted to Metasploit and made nineteen failed proof-of-vuln attempts before switching to a direct HTTP request, misattributed the system weakness to an incorrect CVE, and scanned two subnets outside its authorized scope during reconnaissance. Roughly half of its tool calls during the engagement were exact duplicates of a previous call.

When pitted against Grafana, the agent was tested across all four assistance levels. Without guidance, and again with only the class named, the agent correctly identified Grafana as a target and authenticated to it but pursued an API-based attack chain and did not find the path traversal issue. Given the specific vector, it pivoted toward a server-side request forgery approach and still missed the traversal. Only when given the exact payload did it succeed immediately, at which point it also independently extended the attack to pull additional configuration data beyond what the payload alone required.

5.3 Preliminary Findings

These early results suggest several things worth carrying into the framework's continued development. The agent reliably identifies issues and reasons correctly about its class, but without exact payload information it does not reliably construct the precise test required to succeed, which points toward RAG as a meaningful next step rather than a "nice-to-have." The agent also is not reliably constrained to its authorized scope on its own, which suggests scope enforcement needs to be addressed as a framework-level control rather than left to prompting alone.

Tool selection efficiency, including the duplicate call rate and the default preference for Metasploit over a more direct approach, is measurable and improvable, and is exactly the kind of inefficiency the Recommendations pipeline described in Section 4 is designed to surface and act on. Separately, in both engagements the agent identified valid security findings beyond the specific issue it was pointed at, suggesting reconnaissance value independent of whether the primary proof-of-vuln attempt itself succeeds.

Firewall and NAT configuration also proved to be a meaningful source of added complexity and is expected to compound further as testing extends into the more

heavily segmented scenarios described in Section 2. Configuring realistic router behavior surfaced a non-obvious requirement early in this work. A port forwarding rule alone is not sufficient to expose a service through a router. Traffic reaching a forwarded port is silently dropped unless a matching forwarding rule is also added to allow that traffic through, since inbound address translation and the decision to pass that traffic along are enforced separately. This cost significant debugging time before being identified and is now documented and applied consistently across scenarios. Rule ordering introduced a similar class of problems because firewall rules are evaluated in sequence; a broad blocking rule placed before a specific allow rule can unintentionally block traffic that was meant to be permitted, requiring deliberate ordering rather than an arbitrary rule list. These findings suggest that later scenarios, which layer inter-subnet access control and stricter default-deny policy on top of the same challenge present in Scenario 1, will likely demand not just more capable testing but more careful reconnaissance of the network boundary itself (including recognizing what is being filtered and why) before a proof-of-vuln attempt can even begin.

These findings come from a small number of engagements against two of the seven confirmed issues and should be read as preliminary and illustrative of where the framework's optimization work is targeted, not as a statistically representative evaluation of agent performance across the full scenario range described in Section 2.

6. Future Work

Several directions remain for extending this work beyond what has been demonstrated so far. Some work involves completing infrastructure already underway, while other work involves capabilities identified as gaps during preliminary testing and development. Future work includes the following efforts:

- Complete firewall and NAT configuration for every scenario, following the same approach confirmed in Scenario 1, and finish implementing auto bash commands to simulate live ongoing network traffic.
- Develop and confirm additional chained, multistep execution, extending beyond single-host compromise into a fully documented lateral movement path across the attack chain described in Section 2.
- Integrate the CVE-CWE-CIS database developed alongside this research as a RAG source and rerun the Struts2 and Grafana evaluations at each assistance level with this integration in place, allowing a direct, measured comparison of agent performance before and after RAG support.

- Add a secure Import and Export mechanism for generated tools and playbooks, allowing them to be shared between engagements or installations without also exposing the full engagement data currently embedded in each asset's provenance record.
- Extend the Recommendations and Plugins system to the terminal CLI, which already supports the platform's other core capabilities, bringing the two interfaces to parity.
- Add a validation step that confirms a generated tool functions as intended before it becomes available for use in a live session, reducing the risk of a future engagement relying on a broken or ineffective tool.
- Expand preliminary testing into a fuller evaluation across the complete scenario range, including a formal comparison structured across assistance level and RAG condition, to measure the framework's optimization impact directly rather than from a small set of illustrative engagements.

7. Conclusion

This report documents the extension of Cyber-Agent-Flow from an instrumentation platform that observes and reports on autonomous cybersecurity agent behavior into a system that supports iterative, analyst-supervised improvement of that behavior over time. Building on the reproducible cyber range and execution trace instrumentation established in prior work, this report introduced a complete Recommendations and Plugins pipeline that turns post-engagement findings into reusable capabilities. An analyst can now review what an automated analysis surfaced, correct or scope it with their own notes, and approve the generation of a new tool or playbook, which then becomes available to a future engagement, with every generated capability traceable back to the engagement and human decision that produced it.

Preliminary testing against the developed scenarios demonstrated that Cyber-Agent-Flow can reliably identify vulnerable services and reason correctly about the class of issues present, while also surfacing concrete, measurable inefficiencies in how it operates, including redundant tool calls, scope violations, and a reliance on exact payload information to complete reliably. These findings, while limited in scope, point directly at where this system's continued development is targeted, and demonstrate the practical value of the observability this platform was built to provide. Without visibility into how an agent reasons and where it struggles, none of these specific improvement targets would have been identifiable in the first place.

The broader goal of this work is to build a system that learns something specific and traceable from every engagement it runs. This system should also then make that knowledge available to the next one, without altering the underlying language model and without removing the human from the loop at any step.

The work outlined in Section 6 extends this goal further, toward a more complete evaluation across a wider range of realistic network environments and a tighter feedback loop between what an agent gets wrong and what the system does about it. Cyber-Agent-Flow remains, at every stage of this process, a tool meant to augment an analyst's judgment acting as a force multiplier to enable the analyst, rather than replace them.

8. References

Acosta JC, Nazim MT, Guerra T, Du Y, Aggarwal P. Cyber-agent-flow: execution trace instrumentation and analysis for cybersecurity agent workflows (extended abstract). In: Proceedings of the 2026 AAAI Summer Symposium Series; 2026 June 22–24; Seoul, South Korea. Association for the Advancement of Artificial Intelligence. p. 42950. https://ojs.aaai.org/index.php/AAAI-SS/article/view/42950/50510

Anthropic. Cyber competitions. Anthropic Frontier Red Team Research Report; 2025. https://red.anthropic.com/2025/cyber-competitions/

CORE (Common Open Research Emulator) [software repository]. GitHub, Inc.; 2026. https://github.com/raistlinJ/core

Happe A, Cito J. Getting pwn'd by AI: penetration testing with large language models. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023); 2023 Dec 3–9; San Francisco, CA. p. 2082–2086. arXiv:2308.00121.

Happe A, Kaplan A, Cito J. LLMs as hackers: autonomous Linux privilege escalation attacks. Empirical Software Engineering. 2026;31(3):70. arXiv:2310.11409.

Mader R. AI-powered penetration testing tools in 2026. Spark42 Blog; 2026 Jan. https://blog.spark42.tech/ai-powered-penetration-testing-tools-in-2026/

Singer B et al. On the feasibility of using LLMs to execute multistage network attacks. arXiv; 2025. arXiv:2501.16466.

List of Symbols, Abbreviations, and Acronyms

AI	Artificial Intelligence
AJP	Apache JServ Protocol
API	Application Programming Interface
CIT	Collaborative Innovation Testbed
CLI	Command-Line Interface
CORE	Common Open Research Emulator
CVE	Common Vulnerability and Exposure
dig	Domain Information Groper
DMZ	Demilitarized Zone
GGUF	GPT-Generated Unified Format
GPG	GNU Privacy Guard
GPT	Generated Pretrained Transformer
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
NAT	Network Address Translation
OGNL	Object-Graph Navigation Language
RAG	Retrieval-Augmented Generation
RCE	Remote Code Execution
VM	Virtual Machine
Vuln	Vulnerability

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CB TECH LIB