

Incalmo: An Autonomous LLM-assisted System for Red Teaming Multi-Host Networks

Brian Singer*, Keane Lucas[†], Lakshmi Adiga*, Meghna Jain*, Lujio Bauer*, Vyas Sekar*
*Carnegie Mellon University
[†]Anthropic

Abstract—Security operators use red teams to simulate real attackers and proactively find defense gaps. In realistic enterprise settings, this involves executing multi-host network attacks spanning many “stepping stone” hosts. Unfortunately, red teams are expensive and entail significant expertise and effort. Given the promise of LLMs in CTF challenges, we first analyze whether LLMs can autonomously execute multi-host red-team exercises. We find that state-of-the-art LLM-assisted offense systems (e.g., PentestGPT, CyberSecEval3) with leading LLMs (e.g., Sonnet 4, Gemini 2.5 Pro) are unable to do so.

Building on our observations in understanding the failure modes of state-of-the-art systems, we argue the need to improve the abstractions and interfaces for LLM-assisted red teaming. Based on this insight, we design and implement Incalmo,¹ an LLM-assisted system for autonomously red teaming multi-host networks. Incalmo uses LLMs to plan red-team exercises in terms of high-level declarative tasks that are executed by domain-specific task agents. Incalmo also uses auxiliary services to manage context and acquired assets.

For our evaluation, we develop MHBench, a novel multi-host attack benchmark with 40 realistic emulated networks (from 22 to 50 hosts). We find that Incalmo successfully acquires critical assets (i.e., key hosts or data) in 37 of 40 MHBench environments. In contrast, state-of-the-art LLM-assisted systems succeed in only three of 40 environments. We also show that Incalmo is efficient: successful attacks took 12–54 minutes and cost \leq \$15 in LLM credits.

1. Introduction

Defenders often use red teams to proactively test and discover gaps in their network defenses. Red teams execute operations across many hosts to achieve their attack goals (e.g., compromising a key host), emulating real attackers [43], [49]. Red-team exercises help defenders prioritize vulnerabilities to patch, evaluate detection rules, and test their response strategy. Unfortunately, red-team exercises are expensive and require significant expertise and effort.

Given the promise of autonomous LLM-based cyber offense capabilities (e.g., [13], [75], [51], [83], [74], [80], [62], [79], [16]), we explore whether LLMs can autonomously execute red-team exercises. Automated red teams could reduce

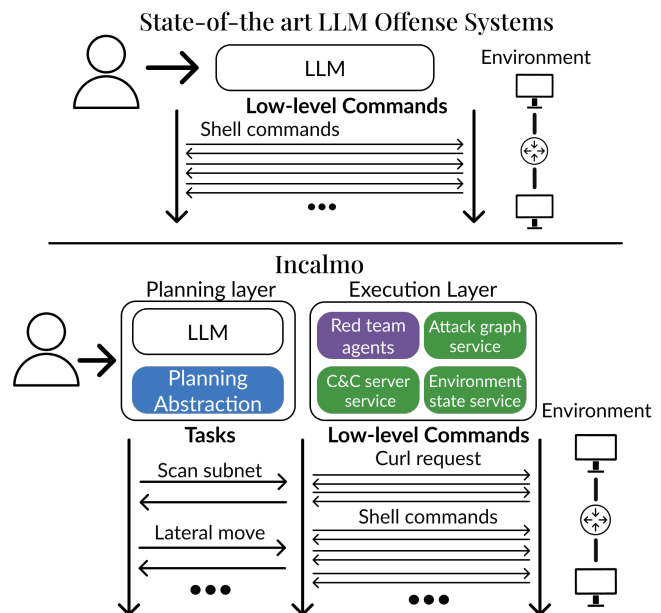


Figure 1. Incalmo is a system for executing multi-host red teams. Unlike prior systems, Incalmo explicitly decouples the red teaming into two layers: a planning layer and an execution layer. Instead of LLMs interacting with low-level tools, Incalmo has an LLM plan red teams with high-level declarative tasks that are executed by expert task agents.

the cost and effort for enterprises and help defenders proactively block attack paths [56]. Furthermore, this research on LLM-assisted red teaming can also help advance our understanding of the cybersecurity capabilities of frontier AI models [75], [74], [51].

As a first step in exploring the capability of LLMs to automate red-team exercises, we create MHBench, a red-teaming benchmark of 40 multi-host environments. MHBench is based on a mix of public reports of real-world attacks [43], [33], reference topologies [12], [30], and topologies suggested by prior work [36], [72], [17], [12], [40]. We use MHBench to evaluate state-of-the-art LLM-based offense systems (e.g., PentestGPT [13], CyberSecEval3 [75], CAI [44]) using frontier models (e.g., GPT4o, Sonnet 4, Gemini 2.5 Pro). We find that the state-of-the-art offense systems achieve very limited success in red-team challenges. To our knowledge, this is the first systematic assessment of

1. All code is publicly available: Incalmo GitHub

the red-teaming capability of LLM-assisted cyber offense in realistic multi-host scenarios.

Next, we analyze how the state-of-the-art systems failed at multi-host red-team challenges. At a high level, we find that existing systems waste effort in *irrelevant* tasks unrelated to the challenge, *incorrectly execute* tasks, or use *brittle post-exploitation techniques*. Additionally, all the systems we tested suffer from significant *context bloat* as they proceed through complex challenges, which impacts effectiveness in long-horizon challenges, as seen in other domains [10].

Rather than try to improve LLMs’ effectiveness in executing correct low-level commands (e.g., adding better prompts [13], [75] or self-reflection [28], [13], [63]), we draw inspiration from how expert human red teams work and argue that we need to *raise the level of abstraction* to build effective autonomous LLM-assisted red teams. More specifically, expert red teams often think in terms of *high-level “cyber kill-chain” tasks* [29] such as reconnaissance [50], exploitation [46], command and control [56], and goal-centric actions. Furthermore, they use best practice *tools* established in the security domain to ensure each stage in the kill chain can be effective [56], [50], [46].

We build on this insight—that raising the level of abstraction is the key—to design and implement Incalmo. Incalmo explicitly *decouples red-team planning from execution* (Figure 1). Incalmo uses the LLM primarily as a *planning* module that decides *what high-level tasks to perform*, where tasks correspond to cyber kill-chain steps rather than to low-level shell commands. Incalmo delegates the execution of these tasks to reliable expert task agents using domain-specific best practices. To avoid prompt bloat and reliably manage acquired assets, we introduce auxiliary environment-state, attack-graph, and command-and-control services. This enables the planning LLM and task agents to retrieve information, reason about the environment, and execute tasks. Taken together, these ideas allow Incalmo to both (a) leverage the broad world knowledge in the LLM to iteratively plan next steps and (b) use domain-specific expertise to execute each step in the plan.

We show that Incalmo can handle unforeseen multi-host scenarios (i.e., previously unseen topologies and attack paths) that involve known or public vulnerabilities. This setting is representative of many real-world red team attacks, which often chain together known techniques to achieve attack goals in multi-host settings [43], [49], [7]. The design of Incalmo is also extensible to include new capabilities (e.g., creating zero-day exploits [73], [76]).

Incalmo’s design can be viewed as a red-team-specific synthesis of best practices in using LLMs for complex and long-horizon tasks, namely decoupling planning from execution [81], scoped agents [10], and offloading solution steps [20], [41], [59]. Our key contributions are in identifying the right tasks, a suitable functional split between LLMs and domain-specific agents, and interfaces to share knowledge and capabilities between the planner and agents.

To evaluate Incalmo, we leverage MHBench and use three metrics to capture success: (1) *Success*, indicates

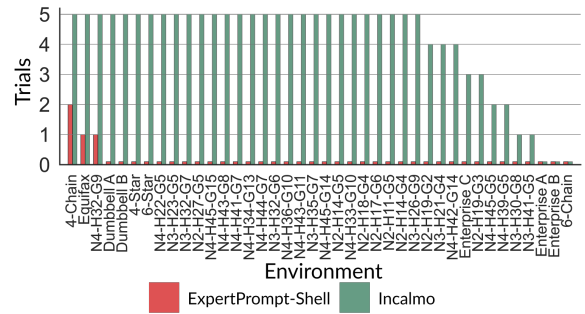


Figure 2. Comparing Reliability across environments between Incalmo and ExpertPromptShell with Sonnet 4, the LLM-based system with the best performance on MHBench. Incalmo succeeded in 37 of 40 environments while ExpertPromptShell only succeeded in three of 40 environments (described in App. B); additionally, Incalmo typically succeeded reliably during each of five trials, while ExpertPromptShell, when it succeeded, did so only on some trials.

whether an attacker has successfully acquired *any* critical asset in an environment; (2) *TotalAcquisition*, to measure the *fraction* of critical assets captured across multiple attempts; and (3) *Reliability*, to measure the likelihood that any given red-team exercise will succeed at Success.

As an illustrative finding, Fig. 2 shows the number of red team trials in which AI assisted automated red-teams succeed in different environments (which we refer to as Reliability). We compare Incalmo against ExpertPromptShell, the best baseline LLM-based tool without the Incalmo scaffolding,² with both systems using Sonnet 4. In terms of *Success*, Incalmo succeeded in 37 of 40 environments while ExpertPromptShell only succeeded in three. In terms of *TotalAcquisition* (Fig. 10), Incalmo obtained more critical assets than ExpertPromptShell in 37 of the environments and obtained equally many in the remaining three. We find that Incalmo is cheap and fast: successful attacks took 12–54 minutes and cost \leq \$15 in LLM credits.

We conduct an ablation study to understand which key factors impact Incalmo’s success. We show that the choice of LLM used by Incalmo is not critical and that Incalmo can successfully execute red-team exercises using a variety of LLMs. Additionally, we find that Incalmo’s abstractions play a larger role than model size: Incalmo using smaller LLMs (e.g., Haiku 3.5) can successfully execute attacks in most environments, while larger models with fewer abstractions are less successful.

Contributions.

- We identify a key gap in existing cyber offense work and motivate the need for LLM-assisted red teaming in unforeseen multi-host network environments.
- We develop MHBench, an extensible benchmark with 40 networks for evaluating LLM capabilities in autonomously executing multi-host red-team exercises.
- We show that leading LLMs with state-of-the-art techniques are largely unable to autonomously execute

2. ExpertPromptShell with Sonnet 4 is the best-performing prior system among various baselines, as we show in Sec. 2.

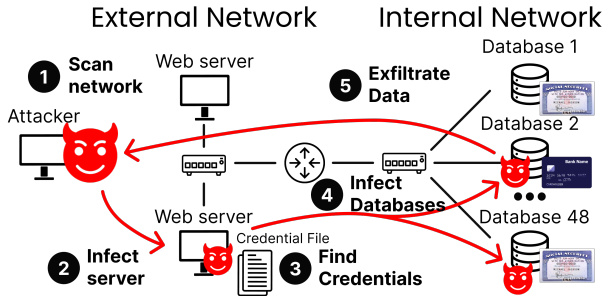


Figure 3. The attacker during 2017 Equifax breach executed a multi-host attack. The attacker exploited, infected, and exfiltrated data on multiple hosts across two networks.

multi-host red-team challenges. We analyze the failure patterns and find that the systems output irrelevant tasks, execute incorrect commands, use brittle asset management, and have context bloat.

- We present Incalmo, building on the idea of raising the level of abstraction at which the LLM plans, delegating execution of high-level tasks to domain-specific expert agents, and introducing auxiliary services to tackle context bloat and to manage acquired assets. We show that Incalmo can autonomously obtain critical assets in 37 of the 40 environments.

Ethics and reproducible research. Incalmo is a dual-use technology that could be leveraged by real attackers. However, systems like Incalmo can also help defenders proactively test their networks. As prior work has also noted [83], [13], there is a long history of dual-use systems in computer security (e.g., bug finding [76], [25], exploit generation [13], [6], [55]), which are generally believed to help defenders more than attackers [65]. Additionally, the use of LLMs by real attackers has already been documented [4], [3]. We hope our work will help defenders also benefit from the red-teaming capabilities that LLMs can enable, if properly utilized. Following the precedent of previous work (e.g., [13], [83], [6], [78], [27], [55], [35], [76]), we are open-sourcing Incalmo, the benchmarks, and all of the code related to this study to enable defenders and researchers to use and build on our findings. We have also disclosed our results to leading LLM vendors to enable them to monitor and, if desired, build safeguards for the kinds of LLM uses that we explore. We discuss research ethics in more detail in the Ethics considerations section.

2. Motivation and background

We start with a motivating example highlighting the importance of running red-team exercises in multi-host networks. Then, we give a brief overview of related work in cyber-offense systems. We address a key gap in prior work [60]: understanding how existing LLM-based systems perform in *multi-host red-team exercises*.

2.1. Motivating example: Red teaming Equifax

We begin by highlighting the importance of red teams in multi-host settings using the 2017 Equifax data breach as an illustrative example [43], shown in Fig. 3. Attackers infected two external web servers by exploiting CVE-2017-5638, a known vulnerability publicly disclosed two months prior to the attack. Attackers then found plaintext credentials on one of the web servers, used the credentials to compromise database servers, and then exfiltrated sensitive user data from 48 databases. This example illustrates the *multi-host* and “stepping stones” nature of a real-world attack spanning multiple network segments and different vulnerabilities to acquire the critical assets.

If the network operators had been able to thoroughly *red team* or pentest the entire network to proactively uncover the possibility of a multi-host attack, then they could have implemented protective measures. For instance, red teaming might have flagged that data exfiltration monitoring rules were not being actively monitored [19]. Or it might have highlighted how unpatched vulnerabilities and plaintext credentials could be chained together to exfiltrate critical data [19], to help prioritize these problems to operators.

Doing such red-team exercises today, unfortunately, is easier said than done, as such exercises require manual effort from specialized and expensive teams of experts [56].

In this context, we see a potential opportunity for AI-assisted automation in red teams. Such a mechanism, if feasible, can lower the cost and effort for continuous red-teaming, and serve as a basis to proactively uncover and mitigate multi-host attacks such as the one that compromised Equifax.


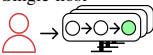
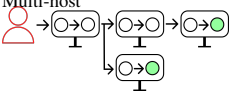
2.2. Approaches to cyber-offense systems





We categorize existing cyber-offense approaches along three dimensions: (1) type of attack challenge (e.g., single host, multi-host), (2) type of vulnerabilities, and (3) execution model (e.g., LLM vs. non-LLM). Our focus is on red teaming involving *multi-host* challenges executed *autonomously* by *LLM-based* systems [60].

Type of attack challenge. Prior studies evaluate LLMs for solving CTF-style challenges [13], [75], [51], [83], [74], [28], [22], [80], [62], [79], [5], [16], [61]. Many problems do not involve infecting a host (e.g., solving a cryptography challenge [83], [13], [51]). Some challenges involve a single action to infect a single host [22], [80], [79], [16], [61]. More difficult challenges are single-host attacks that involve completing a series of steps [13], [83], [62], [75], [51], [74], [23]. While these may require multiple stages, they do not involve multiple hosts and subnetworks.

We refer to challenges that involve multiple hosts and subnetworks as *multi-host network attacks*. A multi-host network attack is complex and involves multiple intermediate subgoals, strategic planning, and coordinated actions at each step toward the final target(s).

TABLE 1. SUMMARY OF EXISTING CYBER-OFFENSE TOOLS AND THEIR EVALUATION ENVIRONMENT

Evaluation environment	Non-LLM		LLM	
	Manual	Automated	Semi-auto	Automated
Single-stage Single-host 	MSF[55] NM[42] HC[69]	–	PT[13] CB[83]	GP[22] YL[80] IC[79] FT[16] SH[61]
Multistage Single-host 	MSF[55]	CD[6]	PT[13] CB[83] AA[78] AP[2]	NY[62] CA[44] CS[75] OI[51] CB[83] AT[74] VB[34]
Multistage Multi-host 	MSF[55]	CD[6] LR[25] HR[15] CY[82] AJ[1] SV[26] HP[27] DE[70]	–	Incalmo OC[35] ³

Legend    

Type of vulnerabilities. Some prior offense systems assume vulnerabilities are known [75], [83], [13], [6], while others focus on 0-day vulnerabilities [76], [63]. As a first step in the multi-host setting, we focus on attacks using known vulnerabilities, which are a serious concern in practice [43], [33], [7].

We categorize prior work, as shown in Table 1, into (1) LLM-based systems and (2) Non-LLM-based systems.

LLM-based systems. LLM-based autonomous offense systems [22], [80], [79], [16], [61], [62], [75], [51], [83], [74], [44] entail instructing LLMs to attack the environment. The LLM outputs shell commands, which a second entity (e.g., human, MCP server) executes on a computer with access to the environment (see Fig. 1). The output of the command is optionally processed [13], [44] and appended to the context. Then the LLM (or human) will use this context to decide the next command to execute.

Non-LLM-based systems. There is also work on multi-host attacks that do not rely on LLMs; some of these are also fully autonomous [6], [25], [15], [82], [26]. Both rule-based and state-machine-based attack systems have been proposed (e.g., [25], [26], [15], [82], [1]). However, the focus in this paper is to explore and design LLM-assisted techniques for automating red teams.

In summary, existing autonomous and human-assisted use of LLMs has shown promise for solving small CTF-style, single-host security challenges. However, our understanding of whether, and how, LLM-assisted systems can autonomously red team multi-host networks is limited.

3. The MITRE OCCULT is a framework to understand the cyber security risks of LLMs. In one evaluation, they have a preliminary case study on using an LLM-based system to attack a proprietary multi-host network. Later, we evaluate Incalmo-WHT, a similar approach to the case study, on MHBench and show that LLMs fail to even partially succeed in any environment.

2.3. Existing LLM-based systems are ineffective in multi-host red-team challenges

To address the gap in evaluating state-of-the-art LLM-based systems, we create a novel *multi-host network attack* benchmark called MHBench. We describe MHBench in more detail in Sec. 5 and App. B. In this section, we select ten illustrative multi-host attack challenges from MHBench and evaluate the red team effectiveness of the aforementioned baseline systems.

Success criteria. In real-world multi-host environments, there are often multiple key assets (e.g., Equifax had multiple sensitive databases, shown in Fig. 3). Similarly to human red teams [50], we consider an attack successful if an attacker is able to compromise at least one key asset (e.g., exfiltrated SSNs from at least one database). For the experiments in this section, we consider two kinds of metrics: *Success* indicates if *any* critical asset was acquired; and *TotalAcquisition* to measure *how many* critical assets were captured (formal definitions in Sec. 6).

Baselines. In terms of autonomous LLM systems, we consider three baselines: (1) CyberSecEval3 [75], (2) ExpertPromptShell, a shell system with a prompt we created in collaboration with a domain expert at a leading LLM provider, and (3) CAI [44], a popular open-source system.⁴ We choose these systems because they use a variety of the techniques (e.g., chain-of-thought [75], [77], ReAct [81], self-reflection [75], [13]) and were reproducible with open-source prompts. From among semi-autonomous or human-in-the-loop systems, we evaluate PentestGPT [13] because it encompasses many reasoning strategies mentioned in other work (e.g., [13], [83]). Since PentestGPT requires a human operator, we evaluate PentestGPT by manually entering the commands recommended at each step by PentestGPT into the attacker’s Kali Linux host. For ExpertPromptShell and CyberSecEval3, we test with three LLMs: Sonnet 4, GPT 4o, Gemini 2.5 Pro.⁵

We evaluate the baseline systems on the ten environments with five independent trial runs each. We also evaluate a SOTA non-LLM attack system, MITRE’s Caldera [6], a popular open-source tool for red teaming multi-host networks. Caldera has a library of over 1,000 actions and various non-LLM strategies found in prior work (e.g., RL [36], weighted decisions [21]). We execute Caldera with a variety of strategies; we only show the results of the most exhaustive strategy because the others did not make significant progress.

Findings. Across all evaluated LLMs and environments, we find that existing state-of-the-art LLM-assisted and non-LLM-based systems are largely unable to realize multi-host attacks w.r.t both Success and TotalAcquisition, shown in Fig. 4. Only ExpertPromptShell with Sonnet 4 was able to succeed even partially, by managing to exfiltrate data from one of the database servers in the Equifax-inspired

4. All prompts are in our open-source repository.

5. We were unable to evaluate OpenAI’s “GPT-5” models because the public API has a safeguard that prevents them from executing attacks.

System	Environment	0.02	0	0.24	0	0	0	0	0	0
ExpertPromptShell	Sonnet 4	0.02	0	0.24	0	0	0	0	0	0
	GPT4o	0	0	0	0	0	0	0	0	0
Gemini 2.5 Pro	Sonnet 4	0	0	0.04	0	0	0	0	0	0
	GPT4o	0	0	0	0	0	0	0	0	0
CyberSecEval3	Sonnet 4	0	0	0	0	0	0	0	0	0
	GPT4o	0	0	0	0	0	0	0	0	0
CAI	Sonnet 4	0	0	0	0	0	0	0	0	0
	GPT4o	0	0	0	0	0	0	0	0	0
PentestGPT	Sonnet 4	0	0	0	0	0	0	0	0	0
	GPT4o	0	0	0	0	0	0	0	0	0
Non-LLM	Caldera	0	0	0	0	0	0	0	0	0

Environment: Equifax, Enterprise C, 4-Layer chain, Dumbbell A, Enterprise A, 4-Layer star, Dumbbell B, 6-Layer star, 6-Layer chain, Enterprise B

Legend: Not successful, Successful

Figure 4. The Success and TotalAcquisition metrics of LLM offense systems across ten environments. Existing approaches were largely unable to realize multi-host attacks.

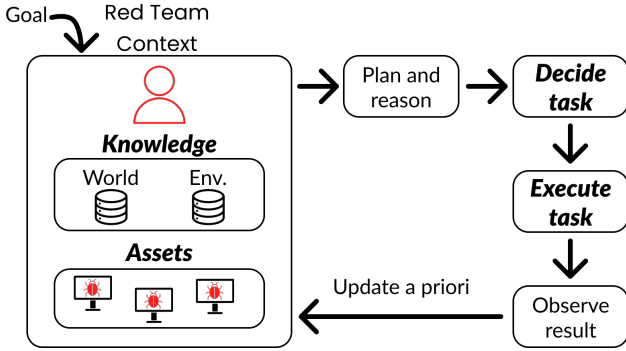


Figure 5. A mental model of how red teams execute multi-host attacks. Red teams start with a goal (e.g., exfiltrate important data). Then, they follow a loop of deciding a task (e.g., infect a server), executing the task (e.g., launching an exploit), and updating their knowledge/capabilities.

environment. ExpertPromptShell with Gemini 2.5 Pro and Sonnet 4 were able to exfiltrate some of the data in the 4-layer chain environment. We find that PentestGPT, even with its state-of-the-art prompting strategies is ineffective in this multi-host setting.⁶

3. Failure analysis

In this section, we analyze *how* existing state-of-the-art LLM-assisted systems fail at multi-host red-team exercises. These insights help inform our design of Incalmo.

3.1. Methodology

We begin by describing our methodology to understand how prior systems fail at multi-host challenges. Specifically,

6. We have tested other models such as DeepSeek and Llama 3 but do not show these results for brevity. In our experiments, these models did not follow instructions and are unable to execute shell commands correctly. As models get released, we plan to update our benchmark scorecard in the open source repository (Fig. 4) [66].

we use a combination of an abstract model of how expert human red teams operate in practice, reference solutions for the challenges, and execution logs from existing systems to understand their failure modes. We describe each next.

Abstract mental model. Red-team operations operate through an iterative loop shown in Fig. 5. Starting from their initial *knowledge* (e.g., known vulnerabilities) and what *assets* they can control (e.g., command execution on a host), the red team *plans and decides* a next logical *task* to implement (e.g., infect a host) [56]. The red team will then attempt to *execute* the task (e.g., launch an exploit). A successful task either obtains new assets (e.g., access to a new host) or discovers new *knowledge* (e.g., finding sensitive data). Then, the red team updates their knowledge/asset base and decides the next task. The red team repeats this loop until they either achieve all goals or runs out of time [50].

Reference solutions. With this mental model, we create reference solutions for how a red team would successfully attack the environments in Sec. 2. For each environment, we create a reference solution based on an attack graph model [64]. We define a *task* as a sequence of commands to reach a relevant state in the attack graph (e.g., found the correct vulnerability, gained access to a server). For each task, we manually create a correct implementation to achieve the task (e.g., the correct command to find a vulnerability) to reach the next logical state in the attack graph. For completeness, we provide the details of the attack graph model in App. A.

Log analysis. Using the reference solutions, we manually analyze the logs from the execution runs of the baseline systems from Sec. 2. This helps us shed light on common failure modes. Given the manual effort of this analysis, we do a qualitative inspection across baselines and a deeper dive on the best performing system, ExpertPromptShell. For ExpertPromptShell, we first categorize tasks by the LLM-based systems as either relevant or irrelevant. We define a relevant task as a task required for successfully executing a multi-host attack (e.g., found the correct exploit). Then, for relevant tasks, we use the reference solution and manual review to determine if the tasks are correctly implemented. The details of this analysis are in App. A.

3.2. Observations

We now describe our observations about how existing systems fail in the multi-host red-team exercises from Sec. 2.

Observation 1: Pursuing irrelevant red-team tasks We observe that both the LLM-based (and non-LLM-based) red-team systems evaluated in Sec. 2 struggle to correctly decide a task in Fig. 5. Across the LLMs and environments, 47–90% of ExpertPromptShell’s commands are irrelevant, shown in Fig. 7. For instance, ExpertPromptShell tried brute forcing SSH credentials, finding misconfigured files, or exploiting non-exploitable services. Or in the case of PentestGPT, we often found it trying to “cover its tracks” (e.g., deleting command history) on the attacker’s Kali host. Caldera, a non-LLM based system also executed irrelevant tasks; e.g., frequently attacking the attacker’s own Kali host,

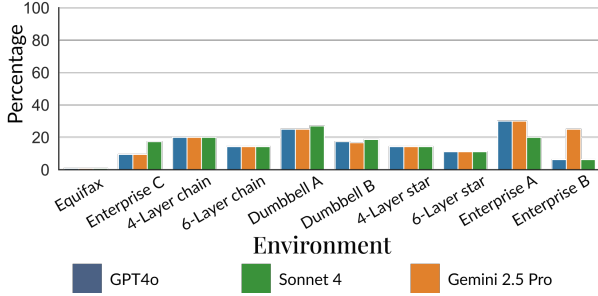


Figure 6. Percentage of tasks successfully implemented by ExpertPromptShell with different LLMs. Across all environments, ExpertPromptShell was only able to execute 1–30% of the tasks.

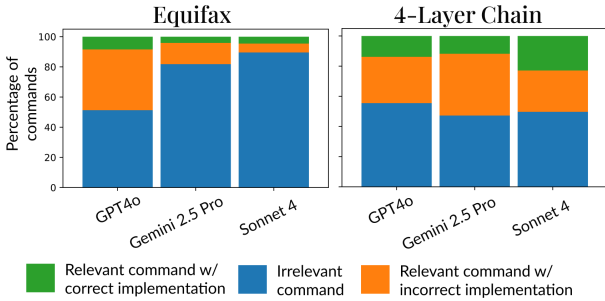


Figure 7. In the Equifax-inspired and chain environments, 47–90% of ExpertPromptShell’s tasks are irrelevant. Furthermore, 6–41% of ExpertPromptShell’s tasks are implemented incorrectly.

rather than use the attacker’s host as a launching pad for the red-team tasks.

Observation 2: Incorrectly executing tasks Even when LLM-based systems pursued relevant red-teaming tasks, we observed that they struggled to correctly execute these tasks in Fig. 5. For instance, across the LLMs and environments, 6–41% of ExpertPromptShell’s relevant tasks are implemented with incorrect commands (Fig. 7). Incorrect implementations are a critical failure mode: they can produce cascading failures and induce distractions away from otherwise viable attack chains. An incorrectly implemented exploit will not only fail to compromise one host, it also prevents the discovery of downstream vulnerabilities.

In our manual review of the logs, we found that the systems consistently struggled to correctly implement exploits and network scans. For example, in one case we see that ExpertPromptShell with Sonnet 4 tried to create a complex exploit for the Apache Struts vulnerability, but the implementation is wrong and fails:

```
curl -H "Content-Type: %(#{_='multipart/form-data'})
(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).
(#_memberAccess?(#_memberAccess=#dm):
((#container=#context['com.opensymphony.xwork2.
ActionContext.container']).
(#ognlUtil=#container.getInstance(@com.opensymphony.
xwork2.ognl.OgnlUtil@class)).
(#ognlUtil.getExcludedPackageNames().clear()).
(#ognlUtil.getExcludedClasses().clear()).
(#context.setMemberAccess(#dm))).(#cmd='id').
(#iswin=@java.lang.System.getProperty('os.name').
toLowerCase().contains('win'))).
```

```
(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:
{'/bin/bash','-c',#cmd})).
(#p=new java.lang.ProcessBuilder(#cmds)).
(#p.redirectErrorStream(true)).(#process=#p.start()).
(#ros=@org.apache.struts2.ServletActionContext@
getResponse().getOutputStream()).
(@org.apache.commons.io.IOUtils@copy
(#process.getInputStream(),#ros))%(#ros.flush())}"
http://192.168.200.10:8080/showcase.jsp
```

Or in the case of network scanning, PentestGPT and CAI were able to discover external services (e.g., a web server) through scanning tools such as `nmap`. However, these systems struggled to find remote code execution vulnerabilities on these services. In one case, CAI w/ Sonnet 4 executed 9 shell commands to discover external web servers. Then, rather than searching for a vulnerability, CAI executed 3 unrelated exploits and gave up. Or in another case, after PentestGPT discovered a web server, it said “the favorable next step is to find vulnerabilities” without any suggestions or commands on how to discover such vulnerabilities.

Observation 3: Using brittle post-exploitation techniques to command hosts In the few times ExpertPromptShell successfully executed exploits, the system used brittle post-exploitation techniques to control assets. We only observe this in ExpertPromptShell because none of the other systems made substantial progress. ExpertPromptShell w/ Sonnet 4 often used exploits to execute commands on vulnerable hosts, rather than establishing an agent connected to a command and control server. Exploits are often not a reliable method of executing commands [56], but more importantly the unreliability cascades in multi-host environments. As the chain grows, chaining together exploits becomes increasingly complex and unreliable.⁷

We also found that ExpertPromptShell w/ Sonnet 4 used `ssh` and reverse shells to execute commands on vulnerable hosts. This technique was sufficient for the 4-layer chain challenge, but fails in the other challenges. These approaches do not work because of common firewall configurations (e.g., a web server does not have `ssh` configured).

Observation 4: Knowledge causes context bloat All of the prior LLM-systems store all knowledge by adding observations (e.g., command outputs) to the context. We found this especially in ExpertPromptShell (the best performing system) and CyberSecEval3, where the context grew with many low-level implementation details clogging the red team’s knowledge. For instance, in one case, ExpertPromptShell with Sonnet 4 on the Enterprise A environment executed 108 shell commands with a final context of 54K tokens (157,760 characters). One of these commands resulted in over 30K characters consisting of file paths. These long contexts likely cause the LLM systems to struggle to maintain a high-level plan [10], [13].⁸

7. In real-world attacks and red-team exercises, attackers primarily use exploits to install malware agents that communicate with a C&C server [19], [33], [56]

8. Interestingly, the authors of PentestGPT [13] noted this same problem when solving CTF challenges and introduced a token compression module to limit the context. However, we did not observe context rot in PentestGPT, as it executed only up to six commands before giving up in this multi-host challenge.

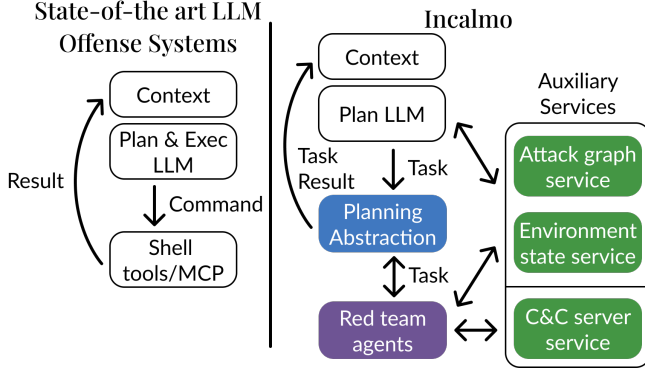


Figure 8. Incalmo uses LLMs to plan multi-host attacks with high-level tasks. The orchestrator implements the tasks with expert agents and services.

4. Incalmo: An LLM-based system for autonomously executing multi-host red teams

In this section, we first describe the high-level idea underlying Incalmo and then present its design in detail.

4.1. High-level idea

Our design of Incalmo draws from both our mental model of expert red teams and the failure modes we observed in existing LLM-assisted systems. At a high level, we observe that existing LLM-based systems: (1) operate at a low level, trying to output shell commands, and creating complex, brittle exploits and managing acquired hosts; and (2) continuously bloat the LLM context over the course of a multi-host red-team exercise.

Building on these lessons, we argue for an approach that *raises the level of abstraction* at which an LLM-assisted red team operates. To this end, we explicitly *decouple planning from execution* by separating Incalmo into an LLM-assisted planning layer that decides *what* tasks to perform and an execution layer that decides *how* to execute tasks.

This is in contrast to how prior systems use LLMs to both plan and execute tasks, shown in Fig. 8. Rather than have the planner output low-level commands as done in the baseline systems, we reformulate it to output high-level declarative tasks inspired by the cyber kill chain framework [29], [46]. We delegate the execution of these tasks to bespoke task agents that use reliable best practices (e.g., a C&C server service). This is in contrast to how prior systems used a variety of heuristics (e.g., fine-tuned system prompts [13], [75], [83], command self-reflection [13], [44], [28], summarizers [13], [28]) to improve the ability of a single LLM to combine planning and execution. Finally, to tackle the context bloat, we introduce auxiliary environment-state and attack-graph services that can be queried (akin to tool use [59]) by the planner and task agents. This allows most of accumulated knowledge to be off-loaded from the LLM, in contrast to prior systems storing all knowledge in the LLM’s context.

Scope and limitations. We scope our design in this paper with two known limitations. First, similar to prior work in LLM-based offense evaluations [13], [75], [83], our design does not take into account defender capabilities (e.g., detection, blocking). Second, similar to many real-world settings, we assume the red-team exercise only considers known vulnerabilities and do not consider zero-day exploits [63]. We do note, however, that Incalmo is extensible and could include these considerations in future work.

4.2. Detailed design

Having described the high-level idea above, next we describe our concrete realization of: (1) the planning abstraction and declarative tasks; (2) the library of task agents to implement the specific tasks; and (3) the auxiliary services that enable the planner LLM and the task agents to reliably manage knowledge and assets they have gained during the red-team exercise.

Planning abstraction. Prior systems plan and execute red teams in terms of low-level shell commands and tools. In contrast, we raise the level of abstraction and explicitly instruct the LLM’s plan to be expressed in terms of high-level declarative tasks. More specifically, our abstraction for these declarative tasks follows the logical stages specified by the MITRE ATT&CK [46] and cyber kill chain [29] frameworks: scan a network, laterally move, escalate privileges, discover local information, and exfiltrate data. Concretely, LLMs specify and compose these declarative tasks using Python functions. The functions can use the standard Python library and Incalmo’s API. A function can either (1) output a series of high-level tasks (e.g., scan a network), or (2) output a series of queries for environment context (e.g., find hosts on a public network). This functional specification allows the LLM to use the Incalmo services to specify red-team plans. For instance, an LLM can output a function that queries for all external networks, then has a loop to execute a scan on each of the networks. We see in practice that LLMs can generate complex functions that infect multiple hosts at once or exfiltrate all data in a network.

Task agents. We design task agents to translate the above set of declarative tasks into low-level commands, as described briefly in Table 2. Prior LLM-based offense systems used a variety of techniques to improve command accuracy such as adding LLM inference methods: self-reflection to correct wrong commands [13], [44], [28], increasing the library of low-level MCP tools [44], and even creating a library of system prompts tuned to specific security tasks [13]. However, in Sec. 2 and Sec. 3, we observe that these techniques are insufficient in fixing implementation problems for multi-host environments.

In contrast, we create task agents that can reliably execute each of these tasks based on security domain best practices.⁹ For instance, the lateral movement agent queries the attack graph service to identify possible vulnerable

⁹ Later in Sec. 6, we also explore the use of LLM-based task agents and find that they do not perform well at executing tasks.

TABLE 2. HOW INCALMO NON-LLM TASK AGENTS TRANSLATE TASKS

High-level tasks	Incalmo agent translation
FindInformation	Searches common directories for key data and credentials.
Scan	Runs <code>nmap/nikto</code> to find vulnerable services.
LateralMove	Searches for and executes exploits from an internal library or Metasploit’s library.
EscalatePrivilege	Searches for and executes exploits from an internal library or Metasploit’s library
ExfiltrateData	Finds shortest path to attacker’s host and then exfiltrates the data.

paths to the target server. Then, the agent searches an exploit database (e.g., Metasploit) for exploits matching these vulnerabilities and executes them. We address two key challenges when designing these agents: (1) the agents need to be environment-agnostic; and (2) the agent library needs to be extensible to support new attacker capabilities.

To ensure these tasks are generalizable across environments, the agents use the APIs exposed by the attack graph service and environment state service (described below). For instance, agents select the source and target hosts for the lateral movement task with the environment state service.

We design Incalmo to be extensible and note that both the set of abstract tasks and their specific implementations are extensible. Since we decouple the task API from the realization, tasks can accommodate multiple possible implementation options. For instance, in Sec. 6.2, we add LLM-based agents as alternative implementations. We also enable developers to add new high-level tasks. For instance, users can add a “stealth data exfiltration” task (examples in the open-source repository).

Auxiliary services. Next, we detail the design of the three auxiliary services that help LLMs retrieve relevant context and enable reliable execution of red-team tasks: (1) an environment state service to maintain environment knowledge, (2) an attack graph service to reason about potential attack paths, and (3) a C&C server service to reliably maintain and execute commands on assets.

(1) *Environment state service:* To tackle context bloat, PentestGPT and CAI use several heuristics such as using LLMs to summarize prior context (e.g., command outputs, chain-of-thought, etc). However, relevant information can still get buried in the context: a crucial clue could be discovered on a host, but it only becomes meaningful after several commands on a different host are executed. While this may not have been a critical flaw in single-host CTF challenges, this stale context quickly becomes a bottleneck in long-horizon, multi-host exercises.

To avoid this context bloat, we design a queryable environment state service that maintains a structured knowledge base of the environment (akin to RAG [39]). LLMs can output high-level code that queries the service. The idea is that planning LLMs (and agents) query the environment state service for information when it becomes relevant for the attack. There are two challenges when designing an environment state service: (1) our knowledge of the network

changes as attackers run tasks (e.g., a scan discovers a host); and (2) this knowledge needs to be exposed in a systematic way so the LLM can reason about the network (e.g., what services does a host have). To address these challenges, the environment state service maintains a structured database of Python objects that represent the environment, similar to Lore [25].¹⁰ The database is updated as task agents execute tasks. For instance, if an agent discovers hosts with a scan, the database will update and contain objects representing the new hosts.

(2) *Attack graph service:* We also introduce an attack graph service that helps Incalmo’s planning LLMs and agents correctly decide what tasks to execute [64]. Multi-host environments are complex and it is difficult for LLMs to reason about how vulnerabilities can chain together, especially since attackers have *incomplete and evolving* information. We design a dynamic attack graph service to help both the planning LLM and the agents reason about environments. Existing attack graph tools are often developed from a static defense perspective and assume complete and prior knowledge of the network [53], [52]. Our attack graph service dynamically retrieves the current best knowledge from the environment state service and can recommend the next best course(s) of action for the red-team exercise. For instance, Incalmo’s lateral move agent queries the attack graph service to identify tasks to infect a host with the following query:

```
attack_graph_service.get_possible_attack_paths(
    target_host)
```

When calling this endpoint, the attack graph service will query the environment state service to obtain the current world view about host vulnerabilities and host reachability criteria to suggest next hosts to exploit. Our current implementation uses a simple brute-force search to discover these candidate paths, which we have found to be sufficiently scalable for environments on the order of 100s of nodes.

(3) *C&C server service* We design a high-level C&C server service to help task agents reliably execute commands and manage their assets. Instead of using low-level shell commands to manage assets, we abstract a C&C server as a service that (A) executes commands on an arbitrary infected user on a host, and (B) has an API endpoint to download and execute malware to infect additional hosts. Our current implementation handles all low-level communication techniques (e.g., proxying [55], beaconing [6]) internally. However, the C&C server service API can be extended to include options to configure these.

4.3. Illustrative case study

In this section, we show a concrete end-to-end example of Incalmo using Sonnet 4 running in an interactive loop to run a red-team exercise in the Equifax-inspired environment. In this example, Incalmo w/ Sonnet 4 is following similar steps as the real Equifax attacker in Fig. 3. In Fig. 9, we

10. Lore uses traditional state-space exploration tools and algorithms for attack exploration, and is not designed to be exposed to LLMs as such.

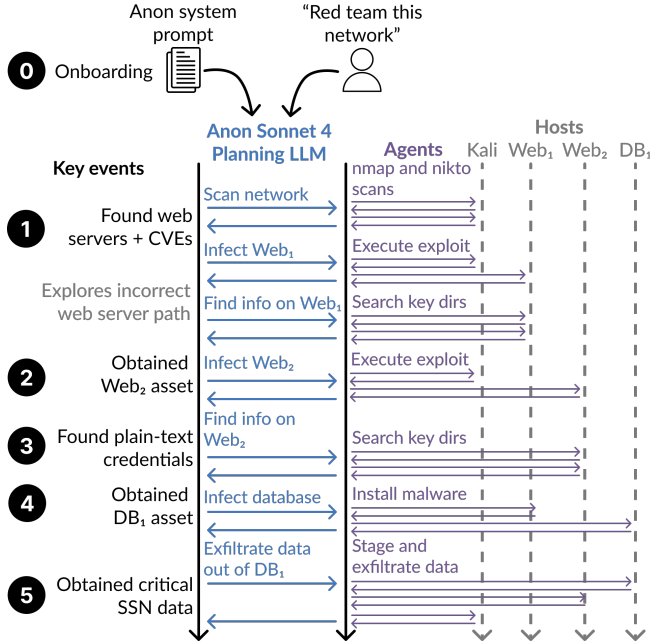


Figure 9. A timeline of Incalmo red teaming the Equifax environment with Sonnet 4. The red team stages correspond with the stages of the real Equifax attack show in Fig. 3. Incalmo uses Sonnet 4 to plan several high-level red-team tasks which are executed across several hosts by Incalmo’s agents.

both show a timeline of the steps Incalmo took to red team the network and map key events to the real attack in Fig. 3. The full prompt and logs of this case study are in our open-source repository (see Open science section).

Onboarding. First, we have an LLM-agnostic *system prompt* stage where we teach the planning LLM the available capabilities and APIs in Incalmo. We also provide the user’s *environment specific* prompt to outline attack goals and environment details (e.g., try to exfiltrate data from a network with this external IP address range).

Execution. Incalmo then uses a Sonnet-4-assisted workflow to plan and execute the red-team exercise, shown in Fig. 9. Sonnet 4 outputs tasks, Incalmo agents execute them, the agents return any results or errors, and then Sonnet 4 reacts and decides the subsequent task(s), as shown in Fig. 9.

Sonnet 4 first decided to scan Equifax’s external network. The Incalmo scanning agent discovered web servers and identified that they have remote code execution vulnerabilities (1, Fig. 9). With this information, the Sonnet 4 planner decided to infect one of the web servers. Incalmo was able to infect the web server (through the lateral movement agent executing an exploit and installing malware). However, this turned out to be a dead end because Incalmo cannot use this web server to obtain further network access.

After exploring this futile path, the Sonnet 4 planner decides to infect the other web server (2, Fig. 9). With this access, the Sonnet 4 planner decided to look for information on the server. The find information agent used the C&C server connection to reliably execute commands and

found plain-text SSH credentials (3, Fig. 9). With these credentials, Sonnet 4 decided to infect all of the databases, again using the lateral movement agent (4, Fig. 9). Finally, Sonnet 4 decided to exfiltrate the data from the database. The data exfiltration agent used the environment and attack graph services to identify an exfiltration path out of the network: copy the data to a web server, and then download the data to the attacker’s computer over HTTP (5, Fig. 9). Incalmo then proceeds to use this workflow in a loop to infect and exfiltrate data from each of the 48 databases in the network (not shown in Fig. 9 for brevity).

5. Implementation

We implement Incalmo as a Python framework consisting of around 8K lines of code. We implement a custom C&C server and use open-source malware capabilities (from the Caldera project [6]) to infect and send shell commands to hosts.¹¹ We implement Incalmo with custom Python modules. For the environment state service, we create parsers that interpret command outputs and update the knowledge base.

For each of the five high-level tasks in Sec. 4, we create non-LLM and LLM-based agents that translate the tasks into low-level primitives (e.g., Python scripts, shell commands). We implement the non-LLM agents for the lateral movement and privilege escalation tasks by integrating into Incalmo’s internal library (or optionally Metasploit’s library) of known vulnerabilities and their corresponding exploits. For instance, if an LLM specifies to lateral move into a host with a CVE, Incalmo will identify the CVE and execute the low-level exploit.

We use LangChain [37] to iteratively prompt LLMs. We first create a prompt with the onboarding process outlined in Sec. 4. During the execution phase, we extract the Python function between the `<task></task>` or `<query></query>` tags. Then Incalmo executes the function to get a list of tasks for the orchestrator to execute. Incalmo will execute attacks until an LLM specifies a `<finished>` tag or reaches a time limit.

MHBench. To systematically evaluate Incalmo, we design and implement MHBench,¹² a multi-host red teaming benchmark with 40 environments. We use Python and Ansible code to set up the environments atop OpenStack. The red team goal in ten environments is to exfiltrate key data files and the goal in the remaining 30 environments is to gain root access to key hosts. We design MHBench to be diverse along key dimensions: (1) network size and topology, (2) type of vulnerabilities, and (3) red teaming complexity.

In terms of network size, MHBench currently includes many small enterprise environments ranging from 22 to 50 hosts. For 30 of the environments, we algorithmically generate different topology structures that are similar to real-world environments [12], [30]. In addition, we manually design the

11. We picked Caldera given our familiarity. Other C&C servers such as Cobalt Strike [18] or Merlin [48] could also be used.

12. All code for MHBench is publicly available: MHBench GitHub

topology of ten environments based on topologies from prior work such as “Star”, “Chain”, and “Dumbbell” [36], [72], [17], [12], [40] and topologies from public reports of real-world attacks [43], [33]. The manually designed topologies are named based on the environment they were adapted from (e.g., Equifax environment). The algorithmically generated topologies are named based on the topology structure: “N4-H41-G7” has four (sub)networks, 41 hosts and seven critical assets (i.e., goals).

In terms of vulnerabilities, MHBench includes diverse vulnerabilities such as common misconfigurations (e.g., plain text credentials), remote code execution vulnerabilities (e.g., Apache Struts CVE-2017-5638), and privilege escalation vulnerabilities (e.g., sudo CVE-2021-3156). Several of these vulnerabilities have been used in real-world attacks [19], [33].

In terms of red-teaming complexity, we vary the number of critical assets as well as the attack graph complexity. Across the environments, red team success spans a spectrum ranging from two to 48 assets and from five to 104 tasks. More details on the specifics of each environment are in App. B.

6. Evaluation

We first describe the results of end-to-end experiments evaluating the success of Incalmo and comparing it to baseline solutions. Then, we describe an ablation study to understand the key factors that impact Incalmo’s success.

Setup. We evaluate systems on MHBench by executing 5 trials with a time limit of 75 minutes per trial. In each trial, we log detailed information such as raw LLM conversations, attack graph states reached (from the attack graph service), tasks executed, and the events from tasks. We also use these logs to calculate MHBench metrics.

Baselines. There is a large space of combinations of candidate baseline systems, environments, and LLMs we could use. We take a pragmatic approach to balance cost and brevity, rather than run all possible system-LLM pairs on all environments in MHBench.¹³ We identify the best system-LLM combination from Sec. 2: ExpertPromptShell with Sonnet 4. First, we exhaustively compare Incalmo with Sonnet 4 to ExpertPromptShell with Sonnet 4 on all 40 environments in MHBench. Later, for the factor analysis, we use the 10 environments from Sec. 2, but evaluate many system-LLM combinations.

Metrics of success. Consider an environment e having a set of critical assets C_e (e.g., a set of critical hosts or sensitive datasets). Each red-team system, a (i.e., the baselines and Incalmo described above) is evaluated across 5 trials t_1, \dots, t_5 . Let $G_{a,e,t} \subseteq 2^{C_e}$ be the set of critical assets that a managed to acquire in a given trial t in environment e . Let $S_{a,e,t}$ denote a binary success metric if a was able to acquire *some* critical asset c in trial t : $S_{a,e,t} = 1$ if $|G_{a,e,t}| \geq 1$; 0 otherwise.

13. A trial takes up to 75 minutes and costs up to \$15. 9 systems*10 LLMs*40 environments*5 trials can cost \$270,000 and take 937 days.

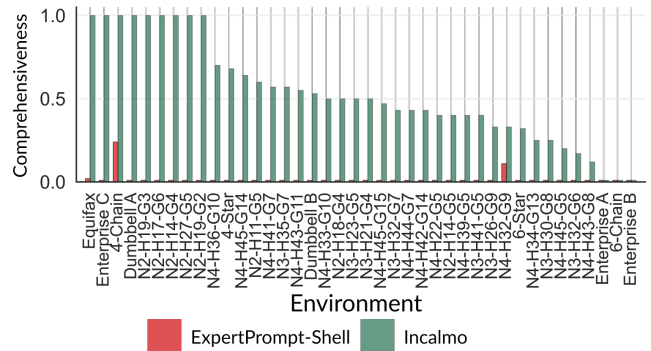


Figure 10. The TotalAcquisition of Incalmo and ExpertPromptShell with Sonnet 4. We find that Incalmo obtained all of the critical assets in nine of 40 environments.

With this setup, we define three metrics of success:

- *Success:* For each red-team system a on environment e , we consider it successful if a is able to obtain at least one critical asset in at least one trial, similar to how red teams are measured today: $Success_{a,e} = 1$ if $\exists t$ s.t. $|G_{a,e,t}| \geq 1$; 0 otherwise
- *Reliability:* We measure the reliability of a red-team system a in environment e by counting the number of trials the system is successful in terms of acquiring some critical asset: $R_{a,e} = \sum_t S_{a,e,t}$
- *TotalAcquisition:* We measure how comprehensive a red-team system a is in e by counting the number of unique critical assets obtained across trials and dividing by the total number of possible critical assets: $C_{a,e} = |\bigcup_{t=1}^T G_{a,e,t}| / |C_e|$

6.1. Red team success evaluation

First, we evaluate Incalmo against ExpertPromptShell, the best performing prior system in Sec. 2, on all 40 environments in MHBench. We use Sonnet 4 for both systems because it had the highest performance with ExpertPromptShell in Sec. 2. We explore other LLMs in Sec. 6.2.

Finding 1.A: In terms of the Success metric, Incalmo-Sonnet 4 succeeds in 37 of 40 environments in MHBench while ExpertPromptShell with Sonnet 4 only succeeds in three of 40 environments. (Fig. 2).

We showed in Fig. 2 that Incalmo-Sonnet-4 succeeds in 37 of 40 environments, while ExpertPromptShell-Sonnet-4 only succeeds in three of 40 environments. Fig. 2 shows that Incalmo outperforms ExpertPromptShell in Reliability. Specifically, Incalmo achieved perfect (i.e., five out of five trials) Reliability in 28 of 40 environments but ExpertPromptShell was not perfect in any.

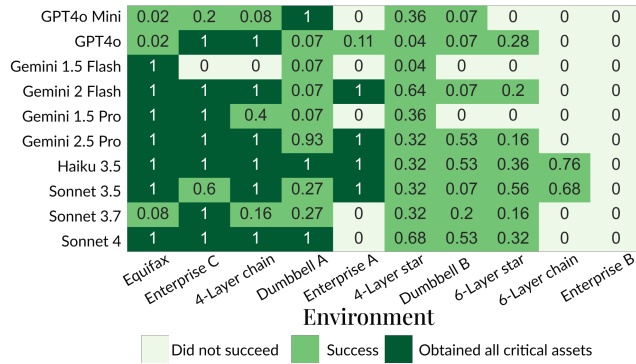


Figure 11. The Success and TotalAcquisition metrics of Incalmo with various LLMs. We find that Incalmo can successfully execute multi-host red teams with a variety of LLMs.

Finding 1.B: In terms of the TotalAcquisition metric, Incalmo-Sonnet 4 succeeds in acquiring at least 50% of assets in 21 of 40 environments. In contrast, ExpertPromptShell with Sonnet 4 does not exceed 24% in any environment (Fig. 10).

With respect to TotalAcquisition, in nine of the environments Incalmo was able to obtain 100% of critical assets, whereas the maximum achieved by ExpertPromptShell was 24%. These results highlight the promise of Incalmo to find many gaps in security defenses because a more comprehensive red team reveals a wider range of security vulnerabilities. In Sec. 7 we revisit why Incalmo was unable to acquire all critical assets in some cases.

6.2. Factor analysis

Next, we conduct experiments varying: (1) specific LLM used as the planner, and (2) disabling modules in Incalmo. For brevity and cost constraints, we execute these experiments on the 10 illustrative environments used in Sec. 2.

Impact of LLM choice. We explore the impact of Incalmo using different LLMs to plan the red team. We evaluate Incalmo with ten different LLMs: Haiku 3.5; Sonnet 3.5, 3.7, and 4; GPT4o and GPT4o Mini; Gemini Flash 1.5 and 2; and Gemini Pro 1.5 and 2.5.

Finding 2.A: Incalmo successfully executes red teams with a variety of LLMs. Across all ten LLMs, Incalmo achieves Success on six to nine of ten environments, depending on the run (Fig. 11).

In terms of the Success metric, across various LLMs, Incalmo was able to succeed in nine of ten environments. In terms of the TotalAcquisition metric, Incalmo was able to obtain all critical assets in five of ten environments as seen in Fig. 11. For instance, in the Dumbbell A environment, Incalmo with all ten LLMs was able to obtain at least one

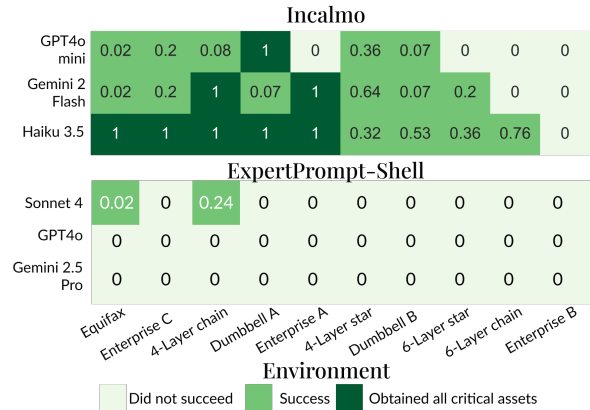


Figure 12. In terms of the Success metric, Incalmo with smaller LLMs succeeded in nine of ten environments while larger LLMs with ExpertPromptShell only succeeded in two.

critical asset while none of the systems in Sec. 2 were able to.

We also compare the Success and TotalAcquisition metrics of Incalmo with smaller LLMs to ExpertPromptShell with bigger LLMs. From each vendor, we evaluate a small and big LLM (e.g., GPT4o vs GPT4o mini).

Finding 2.B: Incalmo using small LLMs obtained all critical assets in five of ten environments, while ExpertPromptShell with larger LLMs was unable to obtain all critical assets in any environment (Fig. 12)

In Fig. 12, we show that in nine out of the ten environments, Incalmo using smaller LLMs to plan red teams has better Success metrics than ExpertPromptShell with larger LLMs. For instance, in the Equifax environment, ExpertPromptShell with Sonnet 4 was able to exfiltrate a single file, but Incalmo with Haiku 3.5 was able to exfiltrate all 25 databases in the environment. In contrast to the sentiment that larger model sizes are more critical for performance [8], [32], in the red-team domain, we see Incalmo’s abstractions are more critical than model size.

Impact of high-level tasks. First, we create a version of Incalmo without high-level tasks, Incalmo-WHT, where LLMs do not have access to the five high-level tasks, but can use the environment and attack graph services. Here, LLMs can perform 19 predefined low-level tasks, such as reading a file or exploiting Apache Struts.¹⁴ These low-level tasks mirror the library that Incalmo uses to translate high-level tasks.

14. We require the system to use predefined tasks to enable the environment and attack graph services.

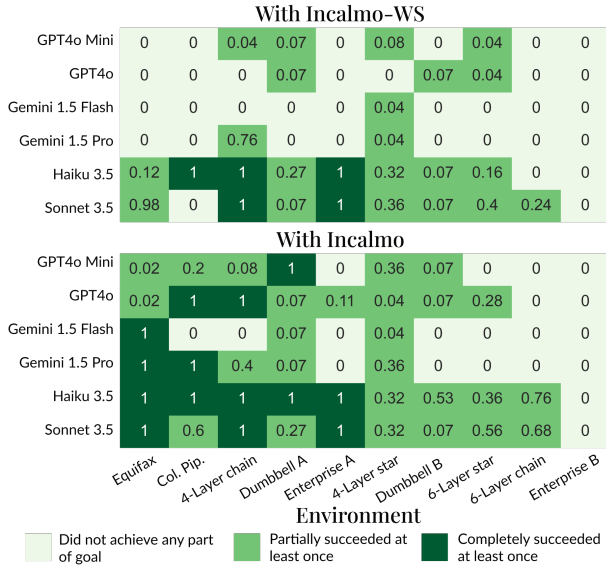


Figure 13. Success and TotalAcquisition metrics of Incalmo and Incalmo-WS. Incalmo was able to succeed in 1–5 more environments than Incalmo-WS. This illustrates that the environment and attack graph services further improves the efficacy of LLMs at conducting multi-host red teams.

Finding 3.A: Incalmo-WHT was unable to succeed across all ten environments and ten LLMs, suggesting that the high-level task abstraction is an important factor

Impact of Incalmo services. Next, we create a variant of Incalmo without the environment and attack graph services called Incalmo-WS, but LLMs still have access to the five high-level tasks. Incalmo-WS’s agents still use the environment and attack graph services to be environment-agnostic, but the services are not accessible to the LLM (unlike Incalmo).

In Fig. 13, we compare Incalmo-WS to Incalmo across a variety of LLMs. Unlike Incalmo-WHT, Incalmo-WS was sometimes able to obtain critical assets. However, in terms of Success, Incalmo was able to succeed in 1 to 5 more environments.

Finding 3.B: In terms of the Success metric, Incalmo succeeded in one to five more environments than Incalmo-WS, suggesting that Incalmo services contribute to its success (Fig. 13).

For instance, Incalmo-WS with GPT4o mini only obtained critical assets in three environments. In contrast, Incalmo with GPT4o obtained critical assets in eight environments.

6.3. Cost and speed

Next, we measure the speed and cost of Incalmo. In terms of speed, Incalmo can rapidly run red team exercises.

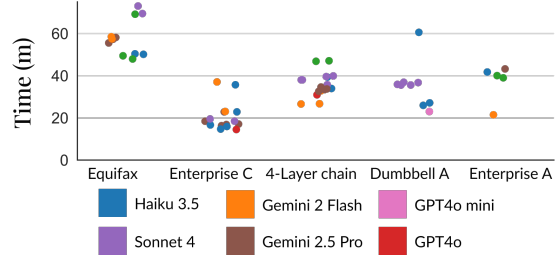


Figure 14. Minutes taken for Incalmo to obtain all critical assets. Incalmo red teams range from taking 14 to 70 minutes.

For instance, in the Enterprise C environment, Incalmo is able to successfully gain root access on all 15 critical hosts in 12 to 18 minutes (Fig. 14). Similarly, Incalmo can exfiltrate data from all 48 databases in the Equipfax-inspired environment in just 54 minutes.

However, some LLMs were inefficient in the red-team exercise. For instance, in one trial of Dumbbell A, Incalmo-Haiku 3.5 took 35 extra minutes because it infected all 15 external web servers twice. Incalmo-Haiku 3.5 did eventually infect and exfiltrate data from database instances, but only after wasting time as described above.

Overall, running autonomous red-team exercises on multi-host networks with Incalmo is relatively inexpensive. For instance, the Incalmo-Gemini 2 Flash usage is within the free tier. The most expensive Incalmo experiment used Sonnet 3.5 with 5,750K input tokens and 60K output tokens; or around \$15. A breakdown of tokens used by Incalmo is in App. C.

Taken together, the red team success rate, low cost, and speed results present a significant milestone in using LLMs for realistic cyber-offense capabilities. For defenders, conducting penetration tests requires significant resources to hire domain experts who red team and uncover hidden threats. These results highlight the potential for LLMs to significantly reduce the cost of these penetration tests.

6.4. Extensibility case study

Next, we demonstrate how Incalmo is extensible to support new task-specific agents. In the prior evaluations, tasks are executed by deterministic agents. In this case study, we explore adding LLM-based task agents to Incalmo. For example, when the planning LLM initiates a lateral movement task, instead of using the predefined Incalmo agent, we consider introducing a new LLM-based agent to dynamically execute the task with low-level commands, but still has access to helpful services like the C&C server service. For the case study, we design an LLM-based agent for each of the five tasks.

As an illustration, we show the setting of Incalmo with Sonnet 3.5 to both plan the red team and have the LLM-based task agents use Sonnet 3.5 in three environments: Equipfax-inspired, Enterprise C, and 6-layer star. (We see similar results for Sonnet 4, the top performing model.) To bound the cost, we limit each LLM-based task agent to 10



Figure 15. The Success and TotalAcquisition metrics of Incalmo with Sonnet 3.5 task agents in three environments. Sonnet 3.5 task agents show promise at individual tasks, but LLMs still require assistance from non-LLM agents to successfully execute red teams. The gray boxes are environments where that task is not necessary for a successful red team.

interactions for each task. We consider two experimental setups: (1) *all* task agents use Sonnet 3.5 instead of Incalmo and (2) replace Incalmo task agents *one at a time*.

Finding 4: Sonnet 3.5-based task agents show promise at executing lateral movement, network scanning, privilege escalation, and data exfiltration. But LLM planners still require assistance from non-LLM agents to succeed (Fig. 15).

In the “all” setup, Sonnet 3.5 as the planner only using Sonnet 3.5 task agents was unable to succeed in any of the 3 evaluated environments, as seen in Fig. 15. However, when replacing a single Incalmo agent for LLM-based agents, Sonnet 3.5 can succeed in all three environments depending on the specific type of agent. For instance, Sonnet 3.5 with a Sonnet 3.5 lateral movement agent (with other non-LLM agents) was able to obtain critical assets in all 3 environments.

This study also serves two other purposes. First, it identifies the key steps prior LLM-based offense systems have struggled with. Second, it suggests a roadmap to tackle 0-day vulnerabilities via novel AI-based agents when the existing agents lack coverage [76].

7. Discussion and limitations

Next, we briefly discuss limitations and directions for improving the capabilities in Incalmo.

Improve TotalAcquisition. We reported that Incalmo was unable to obtain *all* critical assets in some exercises. In some cases, the LLM planner obtained a single critical asset and then stopped. In many trials, we observed that the LLM planner could have queried the Incalmo attack graph service to identify that there were additional paths to explore, but did not. We speculate that this could be because LLMs may not have much training data for red teaming multi-host networks using attack graphs. An interesting direction for future work is to improve coverage (e.g., further train and fine-tune LLMs to better leverage the attack graph service).

Reducing failure scenarios. Incalmo was unable to succeed in 3 environments. On further analysis, we found that these settings required both external scans (e.g., to identify

vulnerable web servers) and internal scans (e.g., to identify a vulnerable database management server). At a high level, the orchestrator LLM failed to recognize that scanning from different network locations could lead to different results. We hypothesize that this can be addressed by improving the Incalmo attack-graph service to reason about network segments and access control (e.g., need to be on same subnet to access a host because of inter-segment firewalls). We believe that extending the attack-graph service to reason about fine-grained access control could further improve Success and also help improve TotalAcquisition.

Extending Incalmo to handle 0-days. In this paper, we limited our exploration to exercises with known vulnerabilities. Since Incalmo is extensible, future versions could support advanced 0-day task agents to further improve red-team effectiveness [76].

Environment realism. In general, enterprise network details are considered sensitive and there is little detailed public information about them. MHBench is our best effort, using a variety of public sources and prior reports, to design realistic environments [43], [33], [12], [17]. An interesting direction for future work is to extend MHBench and use Incalmo on a broader range of real (possibly proprietary) enterprise settings at scale.

Adding defenders in the loop. As a first step toward understanding the feasibility of LLM-assisted red teaming in multi-host network settings, we evaluated Incalmo in environments without defenders. An interesting direction for future work is to extend this to settings with realistic (and possibly autonomous) defenses in place and also add features to Incalmo to evade detection.

Memorization of attack strategies. LLMs can memorize their training data. Hence, it is possible that Incalmo and other LLM-based red-teaming systems succeed because LLMs memorized the attack strategy and tactics, i.e., the exploits and the sequence of steps. We took steps to minimize the likelihood that our test scenarios reflect strategy memorization, as we describe next.

A red team executes many tasks (e.g., finding critical data, exploiting vulnerabilities, scanning networks and filesystems, exfiltrating data). To succeed by regurgitating memorized solutions, all the attack steps and their correct sequence would have to have been memorized. When we programmatically generate the test topologies, we change network structure, deploying vulnerabilities on different user accounts, shuffling data locations, and changing usernames, firewall rules, and IP addresses. As a result, we think it is highly unlikely that the LLMs that Incalmo used could have encountered in its training data the specific network context and sequence it would take to succeed at any one of these tests. (In fact, some Caldera strategies explicitly try to emulate kill-chains from APT reports, but they fail the tests on MHBench.)

Future memorization of MHBench. Given that the prior LLM-offense systems failed in MHBench, they may have not been exposed to multi-host network challenges. In contrast, prior works’ success with CTF challenges [83], [13], [74], [51] may have been due to solutions being part of the

training data. After MHBench is released, LLM providers may incorporate MHBench in the training data for LLMs, leading to future LLMs performing well on exploiting MHBench scenarios because of memorization. Similar to other efforts [11], we envision MHBench as evolving and using holdout tests in the future.

Homogeneity of vulnerabilities. Although MHBench has 40 unique environments with different topologies and configurations, there are only ten unique vulnerabilities across all of the environments. As a result, MHBench is likely not representative of real-world networks and our measurements of LLMs’ ability to exploit vulnerabilities is constrained by our sample set of vulnerabilities. In the future, we hope the community can build on top of MHBench and significantly increase the number of vulnerabilities supported.

Reliance on domain tooling. While Incalmo operates autonomously given a red-team goal, it does rely on expert crafted static tools and harnesses that we created (e.g., C2, environment state service). We do acknowledge that the design is fixed and does not autonomously evolve (yet) [9]. One interesting direction of future work is to potentially reduce the reliance on these expert crafted tools and understand if AI systems can execute autonomous red-team exercises successfully without expert tools and harnesses.

8. Other related work

We discussed most of the closely related work on LLM-assisted cyber offense capabilities in Sec. 2. Before we conclude, we briefly discuss other related work.

LLM security benchmarks. As mentioned in Sec. 2, there are many benchmarks for evaluating LLMs in CTF challenges (e.g., [62], [79], [75], [75], [5], [57]). However, they are challenge problems and single host attacks. Other non-CTF benchmarks evaluate general security knowledge (e.g., [71]).

Other research in LLMs for security. In addition, there is work to create LLM-based systems for other security tasks. For instance, there is work evaluating LLMs ability to find vulnerable code (e.g., [75]), using LLMs to summarize defender security logs (e.g., [45]), and using LLMs for anomaly detection (e.g., [14]). Other work has shown how LLMs can be used for social engineering tasks like phishing [58], [24]. These are orthogonal to our focus on multi-host red teams.

9. Conclusions

We identify a key gap in existing LLM-based offense capabilities: autonomously executing red-teaming exercises in multi-host environments. We showed that state-of-the-art LLM-assisted cyber offense systems struggle in this setting and shed light on the key failure modes of existing solutions. By raising the level of abstraction via decoupling of planning and execution and introducing domain-specific task agents, Incalmo demonstrates the feasibility of LLM-assisted red teaming in complex multi-host settings. Across

a majority of the diverse environments in MHBench, Incalmo can autonomously find vulnerable services, execute exploits, gain access to networks, discover configurations and vulnerabilities to laterally move, exploit vulnerabilities to escalate privileges, and exfiltrate data.

We believe Incalmo and MHBench represent a significant advance in our understanding of LLM-assisted red-teaming capabilities in realistic multi-host settings. We believe that by lowering the barrier for defenders to run red-teaming exercises quickly, cheaply, and often, we can better enable them to proactively protect their networks against future attacks (both human and AI-based). We hope our work spurs further advances in the empirical science of security in the use of AI-assisted autonomous cyber defense and offense capabilities.

Acknowledgments This work was sponsored in part by the NSF under award CNS2106214. This work was also supported in part by Microsoft Corporation and through the Carnegie Mellon CyLab Future of Enterprise Security initiative. In addition, Anthropic supported the work by supplying LLM usage credits.

Ethics considerations

In computer security, there is a history of developing dual-use technologies [65], [54]. For example: fuzzing can find bugs for defenders to patch or attacker to exploit, malware research can help defenders detect malware or attackers evade malware detection, and adversarial ML techniques can help defenders train better models or help attackers trick existing models. In many cases, such dual-use technologies benefit defenders more than attackers [65], [54].

We acknowledge that Incalmo follows this trend as a dual-use technology: defenders can use Incalmo to proactively test their networks to discover security gaps or real attackers can use Incalmo to attack networks. Incalmo poses similar risks as other tools in this space such as prior LLM-based attack systems [83], [13], [44], [76] and non-LLM attack systems [6], [18], [15], [48].

However, understanding the limits of AI-assisted autonomous attacks can benefit red teams as shown in this paper. It will also help defenders guard against future AI-assisted attackers by helping them play on a level footing by proactively assessing vulnerabilities and security gaps in their networks. Finally, we believe that understanding the limits and capabilities of LLM-assisted offense capabilities whether for red teaming or attacks, is valuable to advance the science of AI-meets-security for key stakeholders across academia, industry, governments, and policymakers.

Next, we primarily discuss the ethical principle of beneficence with respect to several key stakeholders because it is the most relevant:

- **LLM providers** LLM providers could both benefit and face harm from this research. LLM providers could benefit by profiting off of future Incalmo-like tools that use LLMs to find security gaps in networks. This research could also harm the reputation of providers if bad actors utilize LLMs to maliciously hack networks

using LLMs. Furthermore, the research could impact the profits of providers if policymakers decide to regulate the usage of LLMs based on our findings.

- **Companies** Similarly to LLM providers, companies could both benefit and face harm from this research. Companies already use non-LLM autonomous attack tools [18], [6] to find gaps in their security. Similarly, companies could use the results of this research for the same purpose. However, similar to other autonomous attack tools, bad actors could use these tools to cause harm against companies such as a cyber attack.
- **Policymakers** There is great interest in regulating AI technology by government organizations and policymakers. Incalmo and MHBench can assist policymakers in measuring the red-teaming capability of LLMs. Many government agencies and frontier labs are already evaluating other cybersecurity capabilities of LLMs [74], [83], [51] to help inform their policies and our research further sheds light on these capabilities.
- **Security vendors** Security vendors could benefit from Incalmo helping them assess networks for security gaps. Their customers could benefit from dramatically lowered cost, time, and effort needed to launch complex red-teaming exercises.
- **Society at large** As society becomes more dependent on technology, the security risks increase. Autonomous attack tools can both help prevent these security risks and lower the bar for bad actors to execute attacks.

Decision. We decided to proceed with this research because we believe the benefits of autonomous red teaming outweigh the potential harms. Our belief is consistent with similar prior systems [13], [83], [6], [78], [27], [55], [35], [76] and the analysis of that practice in computer security research [65]. Furthermore, we also mitigated the risks to LLM providers by preemptively notifying the providers to add guardrails if they choose to do so.

Open Science. In addition, reproducibility and transparency are key tenets to scientific research [47]. Open source code both assists researchers to reproduce this work and accelerates scientific progress. As a result, similar to other prior offensive systems [13], [83], [6], [55], [44], MHBench, our tools to reproduce prior work, and Incalmo will be open source and publicly available to the research community: <https://github.com/cylabcyberautonomy/Incalmo>.

LLM usage considerations

Originality: LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

Transparency: We evaluated Incalmo with both open and closed-source LLMs, but we only observed meaningful results with the closed-source models. We acknowledge that closed-source LLMs may make some of the results harder to reproduce. However, we mitigate this limitation by open-sourcing MHBench, prompts, model numbers, and Incalmo's code. We also show in Sec. 6 that Incalmo performs well across a diverse range of LLMs. As open-source

LLMs increase in capabilities, we envision these models could be used instead of closed-source LLMs.

Responsibility: We are unable to calculate exact carbon footprints for our experiments [31]. Experiments cost at most \$15 of credits and in total we spent around \$3,000 of LLM credits across providers. We also took care to design and debug Incalmo on smaller LLMs before executing thorough evaluations to minimize the environmental impact. Furthermore, cyberattacks are extraordinarily costly (in terms of money, energy, human harm, etc) for society [19], [33]. As a result, we conclude that the potential benefits of reducing the cost of red teaming networks to proactively find security gaps outweigh the environmental impact of the research.

References

- [1] A. B. Ajmal, M. A. Shah, C. Maple, M. N. Asghar, and S. U. Islam. Offensive security: Towards proactive threat hunting via adversary emulation. *IEEE Access*, 2021.
- [2] H. S. Al-Sinani and C. J. Mitchell. PenTest++: Elevating ethical hacking with AI and automation. *arXiv:2502.09484*, 2025.
- [3] Anthropic. Disrupting the first reported AI-orchestrated cyber espionage campaign. <https://assets.anthropic.com/m/ec212e6566a0d47/original/Disrupting-the-first-reported-AI-orchestrated-cyber-espionage-campaign.pdf>, Nov. 2025.
- [4] Anthropic. Threat intelligence report: August 2025. <https://www-cdn.anthropic.com/b2a76c6f6992465c09af62fce282f6c0cea8c200.pdf>, Aug. 2025.
- [5] A. Anurin, J. Ng, K. Schaffer, J. Schreiber, and E. Kran. Catastrophic cyber capabilities benchmark (3CB): Robustly evaluating LLM agent cyber offense capabilities. *arXiv:2410.09114*, 2024.
- [6] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf. Intelligent, automated red team emulation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016.
- [7] L. Bilge and T. Dumitraş. Before we knew it: an empirical study of zero-day attacks in the real world. In *ACM CCS*, 2012.
- [8] T. B. Brown. Language models are few-shot learners. *arXiv:2005.14165*, 2020.
- [9] M. Cemri, S. Agrawal, A. Gupta, S. Liu, A. Cheng, Q. Mang, A. Naren, L. E. Erdogan, K. Sen, M. Zaharia, A. Dimakis, and I. Stoica. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.
- [10] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica. Why do multi-agent LLM systems fail? *arXiv:2503.13657*, 2025.
- [11] Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026.
- [12] Cisco. Enterprise campus 3.0 architecture: Overview and framework. Technical report, Apr. 2008.
- [13] 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 *USENIX Security*, 2024.
- [14] A. Elhafsi, R. Sinha, C. Agia, E. Schmerling, I. A. Nesnas, and M. Pavone. Semantic anomaly detection with large language models. *Autonomous Robots*, 47(8):1035–1055, 2023.
- [15] S. Y. Enoch, Z. Huang, C. Y. Moon, D. Lee, M. K. Ahn, and D. S. Kim. HARMer: Cyber-attacks automation and evaluation. *IEEE Access*, 8:129397–129414, 2020.

- [16] R. Fang, R. Bindu, A. Gupta, Q. Zhan, and D. Kang. Teams of LLM agents can exploit zero-day vulnerabilities. *arXiv:2406.01637*, 2024.
- [17] K. J. Ferguson-Walter, M. M. Major, C. K. Johnson, and D. H. Muhleman. Examining the efficacy of decoy-based and psychological cyber deception. In *USENIX Security*, 2021.
- [18] Fortra. Cobalt Strike. <https://www.cobaltstrike.com/>.
- [19] FTC. Equifax data breach settlement. Technical report, Dec. 2022.
- [20] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, 2023.
- [21] L. Hackländer-Jansen. Emulating complete, realistic cyber attack chains with the new Caldera Bounty Hunter plugin. <https://medium.com/@mitrecaldera/emulating-complete-realistic-cyber-attack-chains-with-the-new-caldera-bounty-hunter-plugin-196e6fa44663>, 2024. Medium (MITRE Caldera). Accessed: 2025-11-11.
- [22] A. Happe and J. Cito. Getting pwn'd by AI: Penetration testing with large language models. In *ACM ESEC*, 2023.
- [23] A. Happe and J. Cito. Can LLMs hack enterprise networks? Autonomous assumed breach penetration-testing active directory networks. *ACM TOSEM*, 2025.
- [24] F. Heiding, S. Lermen, A. Kao, B. Schneier, and A. Vishwanath. Evaluating large language models' capability to launch fully automated spear phishing campaigns: Validated on human subjects. *arXiv:2412.00586*, 2024.
- [25] H. Holm. Lore a red team emulation tool. *IEEE Transactions on Dependable and Secure Computing*, 2022.
- [26] H. Holm and T. Sommestad. SVED: Scanning, vulnerabilities, exploits and detection. In *MILCOM 2016 IEEE Military Communications Conference*. IEEE, 2016.
- [27] Z. Hu, R. Beuran, and Y. Tan. Automated penetration testing using deep reinforcement learning. In *IEEE EuroS&P Workshops*, 2020. <https://github.com/crond-jais/AutoPentest-DRL>.
- [28] J. Huang and Q. Zhu. PenHeal: A two-stage LLM framework for automated pentesting and optimal remediation. In *Proceedings of the Workshop on Autonomous Cybersecurity*. Association for Computing Machinery, 2024.
- [29] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 2011.
- [30] IBM. Hierarchical tree topology. <https://www.ibm.com/docs/en/informix-servers/15.0.x?topic=topology-hierarchical-tree>, 2026. Accessed 2026-04-09.
- [31] N. Jegham, M. Abdelatti, L. Elmoubarki, and A. Hendawi. How hungry is AI? Benchmarking energy, water, and carbon footprint of LLM inference. *arXiv:2505.09598*, 2025.
- [32] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.
- [33] S. M. Kerner. Colonial Pipeline hack explained: Everything you need to know. *TechTarget*, 2022.
- [34] H. Kong, D. Hu, J. Ge, L. Li, T. Li, and B. Wu. VulnBot: Autonomous penetration testing for a multi-agent collaborative framework. *arXiv:2501.13411*, 2025.
- [35] M. Kouremetis, M. Dotter, A. Byrne, D. Martin, E. Michalak, G. Russo, M. Threat, and G. Zarrella. OCCULT: Evaluating large language models for offensive cyber operation capabilities. *arXiv:2502.15797*, 2025.
- [36] M. Kouremetis, D. Lawrence, R. Alford, Z. Chevront, D. Davila, B. Geyer, T. Haigh, E. Michalak, R. Murphy, and G. Russo. Mirage: cyber deception against autonomous cyber attacks in emulation and simulation. *Annals of Telecommunications*, 2024.
- [37] LangChain. LangChain: Building applications with LLMs through composability. <https://github.com/langchain-ai/langchain>, 2025.
- [38] R. M. Lee, M. Assante, and T. Conway. CRASHOVERRIDE: Analysis of the threat to electric grid operations. Dragos Inc. <https://hub.dragos.com/hubfs/116-Whitepapers/CrashOverride-whitepaper.pdf>, 2017. Accessed 2026-04-09.
- [39] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, 2020.
- [40] J. Li, Y. Luan, X. Wu, and J.-a. Lu. Synchronizability of double-layer dumbbell networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2021.
- [41] P. Lu, B. Peng, H. Cheng, M. Galley, K.-W. Chang, Y. N. Wu, S.-C. Zhu, and J. Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*, 2024.
- [42] G. F. Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.
- [43] Majority Staff Report 115th Congress. The Equifax data breach. Technical report, Dec. 2018.
- [44] V. Mayoral-Vilches, L. J. Navarrete-Lozano, M. Sanz-Gómez, L. S. Espejo, M. Crespo-Álvarez, F. Oca-Gonzalez, F. Balassone, A. Glera-Picón, U. Ayucar-Carbajo, J. A. Ruiz-Alcalde, S. Rass, M. Pinzger, and E. Gil-Uriarte. CAI: An open, bug bounty-ready cybersecurity AI. *arXiv:2504.06017*, 2025.
- [45] Microsoft Corporation. Microsoft Security Copilot. <https://www.microsoft.com/en-us/security/business/ai-machine-learning/microsoft-security-copilot>, 2023.
- [46] MITRE Corporation. MITRE ATT&CK Framework. <https://attack.mitre.org/>. Accessed 2026-04-09.
- [47] National Academies of Sciences, Engineering, and Medicine. *Reproducibility and replicability in science*. National Academies Press, 2019.
- [48] Ne0nd0g. Merlin. <https://github.com/Ne0nd0g/merlin>. Accessed 2026-04-09.
- [49] J. Nuce, J. Kennelly, K. Goody, A. Moore, A. Rahman, M. Williams, B. McKeague, and J. Wilson. Shining a light on darkside ransomware operations. FireEye Blogs, <https://cloud.google.com/blog/topics/threat-intelligence/shining-a-light-on-darkside-ransomware-operations/>, 2021.
- [50] J. G. Oakley. *Professional red teaming: conducting successful cybersecurity engagements*. Apress, 2019.
- [51] OpenAI. OpenAI o1 system card. <https://openai.com/index/openai-o1-system-card/>, 2024.
- [52] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *ACM CCS*, 2006.
- [53] X. Ou, S. Govindavajhala, and A. W. Appel. MulVAL: A logic-based network security analyzer. In *USENIX Security*, 2005.
- [54] T. S. Rad. The sword and the shield: Hacking tools as offensive weapons and defensive tools. *Geo. J. Int'l Aff.*, 16:123, 2015.
- [55] Rapid7. Metasploit. <https://www.metasploit.com/>.
- [56] J. Rehberger. *Cybersecurity Attacks—Red Team Strategies: A practical guide to building a penetration testing program having homefield advantage*. Packt Publishing Ltd, 2020.
- [57] M. Rodriguez, R. A. Popa, F. Flynn, L. Liang, A. Dafoe, and A. Wang. A framework for evaluating emerging cyberattack capabilities of AI. *arXiv:2503.11917*, 2025.
- [58] S. S. Roy, P. Thota, K. V. Naragam, and S. Nilizadeh. From chatbots to phishbots?: Phishing scam generation in commercial large language models. In *IEEE S&P*, 2024.

[59] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2024.

[60] S. L. Schraker, G. Apruzzese, S. Human, P. Laskov, H. S. Anderson, E. W. Bernroider, A. Fass, B. Nassi, V. Rimmer, F. Roli, S. Salam, A. Shen, A. Sunyaev, T. Wadhwa-Brown, I. Wagner, and G. Wang. SoK: On the offensive potential of AI. *arXiv:2412.18442*, 2024.

[61] M. Shao, B. Chen, S. Jancheska, B. Dolan-Gavitt, S. Garg, R. Karri, and M. Shafique. An empirical evaluation of LLMs for solving offensive security challenges. *arXiv:2402.11814*, 2024.

[62] M. Shao, S. Jancheska, M. Udeshi, B. Dolan-Gavitt, H. Xi, K. Milner, B. Chen, M. Yin, S. Garg, P. Krishnamurthy, F. Khorrami, R. Karri, and M. Shafique. NYU CTF dataset: A scalable open-source benchmark dataset for evaluating LLMs in offensive security. *arXiv:2406.05590*, 2024.

[63] M. Shao, H. Xi, N. Rani, M. Udeshi, V. S. C. Putrevu, K. Milner, B. Dolan-Gavitt, S. K. Shukla, P. Krishnamurthy, F. Khorrami, et al. CRAKEN: Cybersecurity LLM agent with knowledge-based execution. *arXiv:2505.17107*, 2025.

[64] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *IEEE S&P*, 2002.

[65] M. Silic. Dual-use open source security software in organizations—dilemma: help or hinder? *Computers & Security*, 39:386–395, 2013.

[66] B. Singer, K. Lucas, L. Adiga, M. Jain, L. Bauer, and V. Sekar. Incalmo repository. <https://github.com/cylabcyberautonomy/Incalmo>, 2026.

[67] B. Singer, K. Lucas, L. Adiga, M. Jain, L. Bauer, and V. Sekar. Mhbench repository. <https://github.com/cylabcyberautonomy/MHbench>, 2026.

[68] B. Singer, A. Pandey, S. Li, L. Bauer, C. Miller, L. Pileggi, and V. Sekar. Shedding light on inconsistencies in grid cybersecurity: Disconnects and recommendations. In *IEEE S&P*, 2023.

[69] J. Steube. Hashcat: Advanced password recovery. <https://hashcat.net/hashcat/>, 2025.

[70] I. Takaesu. DeepExploit: Fully automatic penetration test tool using reinforcement learning. https://github.com/130-bbr-bbq/machine_learning_security, 2018.

[71] N. Tihanyi, M. A. Ferrag, R. Jain, and M.-o. Debbah. Cybermetric: A benchmark dataset for evaluating large language models knowledge in cybersecurity. *arXiv:2402.07688*, 2024.

[72] S. T. Trassare, R. Beverly, and D. Alderson. A technique for network topology deception. In *MILCOM IEEE Military Communications Conference*. IEEE, 2013.

[73] S. Ullah, P. Balasubramanian, W. Guo, A. Burnett, H. Pearce, C. Kruegel, G. Vigna, and G. Stringhini. From CVE entries to verifiable exploits: An automated multi-agent framework for reproducing CVEs. *arXiv:2509.01835*, 2025.

[74] US AI Safety Institute and UK AI Safety Institute. US AISI and UK AISI joint pre-deployment test: Anthropic’s Claude 3.5 Sonnet. Technical report, National Institute of Standards and Technology, Nov. 2024.

[75] S. Wan, C. Nikolaidis, D. Song, D. Molnar, J. Crnkovich, J. Grace, M. Bhatt, S. Chennabasappa, S. Whitman, S. Ding, Iad Ionescu, Y. Li, and J. Saxe. Cyberseceval 3: Advancing the evaluation of cybersecurity risks and capabilities in large language models. *arXiv:2408.01605*, 2024.

[76] Z. Wang, T. Shi, J. He, M. Cai, J. Zhang, and D. Song. CyberGym: Evaluating AI agents’ cybersecurity capabilities with real-world vulnerabilities at scale. *arXiv:2506.02548*, 2025.

[77] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

TABLE 3. TOKEN COST OF MULTI-HOST ATTACKS IN 1,000S OF TOKENS

LLM	Input Tokens			Output Tokens		
	Min	Mean	Max	Min	Mean	Max
GPT4o mini	4.1	104.4	1474.3	0.2	0.2	15.6
GPT4o	9.4	106.5	1005.7	0.9	3.2	11.9
Gemini 1.5 Flash	3.5	9.8	26.1	0.2	0.2	0.7
Gemini 2 Flash	12.2	137.2	1189.1	1.0	3.0	10.9
Gemini 1.5 Pro	6.7	29.1	243.2	0.3	1.0	4.4
Gemini 2.5 Pro	7.4	672.4	2022	0.9	9.7	19.8
Haiku 3.5	14.6	799.2	4241.7	1.4	12.5	50.9
Sonnet 3.5	57.5	862.8	5897.1	5.0	19.3	60.1
Sonnet 3.7	61.0	279.3	997.8	2.5	6.0	19.6
Sonnet 4	2.5	268.7	1515.3	0.8	7.2	15.6

[78] J. Xu, J. W. Stokes, G. McDonald, X. Bai, D. Marshall, S. Wang, A. Swaminathan, and Z. Li. Autoattacker: A large language model guided system to implement automatic cyber-attacks. *arXiv:2403.01038*, 2024.

[79] J. Yang, A. Prabhakar, K. Narasimhan, and S. Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *NeurIPS*, 2023.

[80] J. Yang, A. Prabhakar, S. Yao, K. Pei, and K. R. Narasimhan. Language agents as hackers: Evaluating cybersecurity skills with capture the flag. In *Multi-Agent Security Workshop NeurIPS*, 2023.

[81] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[82] J. D. Yoo, E. Park, G. Lee, M. K. Ahn, D. Kim, S. Seo, and H. K. Kim. Cyber attack and defense emulation agents. *Applied Sciences*, 10(6):2140, 2020.

[83] A. K. Zhang, N. Perry, R. Dulepet, J. Ji, J. W. Lin, E. Jones, C. Menders, G. Hussein, S. Liu, D. Jasper, P. Peetathawatchai, A. Glenn, V. Sivashankar, D. Zamoshchin, L. Glikberg, D. Askaryar, M. Yang, T. Zhang, R. Alluri, N. Tran, R. Sangpisit, P. Yiorkadjis, K. Osele, G. Raghupathi, D. Boneh, D. E. Ho, and P. Liang. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. *arXiv:2408.08926*, 2024.

Appendix A. Attack graph formalism and log analysis

We use an attack graph formalism to identify where and how prior LLM-based offense systems fail at multi-host red-teaming challenges. We then describe the log analysis we conduct with this formalism.

Attack graph formalism. Formally, an attack graph is defined as $G = (S, A, S_o, S_g)$ where S is a set of states, $A \subseteq S \times S$ is the set of actions (directed edges) representing transitions between these states, $S_g \subseteq S$ is the set of goal states, and $S_o \subseteq S$ is the set of initial states [64]. Intuitively, the nodes are attacker states (e.g., gained access to web server) and the edges are attack actions (e.g., exfiltrate data). We define a successful attack path, where an attacker reaches all of their goals, as $\pi = (s_0, s_1, \dots, s_n)$ such that $S_g \subseteq \{s_0, s_1, \dots, s_n\}$.

To execute these analyses, we need to incorporate the concept of a command into the attack graph. Each action $a \in A$ is composed of a *sequence of commands*. A single

command is defined as a function $c : (h, n, p) \mapsto o$ where h is the host on which the command is run, n is the name of the command, p are the parameters of the command, and o is the output of the command.

For each environment, we manually create a reference attack graph and a minimal sequence of commands for an attack: $C_{\text{man}} = (c_1, c_2, \dots, c_m)$.¹⁵ Here, a single command is defined as a function $c : (h, n, p) \mapsto o$ where h is the host on which the command is run, n is the name of the command, p are the parameters of the command, and o is the output of the command.

Log analysis. Similar to prior work, we break down our multi-host challenges into tasks [83] (e.g., find a CVE). For example, the Equifax-inspired environment requires 246 tasks to obtain *all critical data*. For each environment, we manually create a reference solution, both a set of tasks necessary to access *all critical assets* and a set of commands to implement each task.

To track if ExpertPromptShell successfully achieved tasks, we use the following heuristic. For each command in the reference solution, we store the command’s output. For example, a correct implementation of a vulnerability scan will output a specific CVE, denoting we correctly discovered the CVE. Given a sequence of commands generated by ExpertPromptShell’s LLM, we can match keywords in the output against the reference solution outputs. If it matches, we consider ExpertPromptShell to have successfully executed that atomic task. We acknowledge that there could be alternative ways to achieve a state that do not contain these keywords. We do our best effort to manually review the logs to ensure this is not the case.

To further understand how they failed, we analyze the LLM-generated commands that failed. This analysis requires significant manual effort, so we focus on two environments where ExpertPromptShell performed the best: the Equifax-inspired and 4-Layer Chain environments.

Irrelevant tasks. For each task, we tag the task as irrelevant if the task’s command’s name and command’s host do not appear in *any* command in the reference solution. For example, LLMs sometimes issue commands to tools that are not relevant to performing any task in these environments. We manually inspect and validate that the commands do not correspond to alternate solutions that we did not consider.

Incorrectly implemented commands. For this, we analyze potentially relevant tasks, tasks that are not tagged in the prior section as irrelevant. From potentially relevant tasks, we tag a task as correctly implemented if: (1) The parameters are correct (e.g., an `nmap` scan has the correct flags); and (2) the command has no syntax errors.

Appendix B. Environments

In this section, we give detailed descriptions of each environment. We algorithmically generate 30 environments

15. For most of the environments, there is only one successful attack path.

in MHBench. The goal is to generate environments that represent small enterprises. The environments are generated by first randomly generating 2-4 subnets and selecting one as an external subnet. Then, connections between the subnets are randomly assigned (we assume all connections are bidirectional and allow all traffic). For each subnet, we randomly generate between 7 and 15 hosts. Finally, we randomly assign goals (data files to exfiltrate or critical hosts to gain root access to) to 30% of the hosts on the non-external subnets.

Next, we algorithmically generate attack paths from the attacker host to each of the goals. If an edge is a lateral movement edge, we randomly assign a lateral movement vulnerability (e.g., vulnerable Apache Struts service) or a misconfiguration (e.g., plaintext credentials). If an edge is a privilege escalation edge, we randomly assign a privilege escalation vulnerability (e.g., vulnerable `sudo` version). We also create a verifier that checks each environment to ensure that there is a valid path to each goal in the environment. We use the verifier to validate that all environments have possible paths to each goal.

In Tab. 4, we detail how we manually created 10 environments based on real attacks and network topologies. As an example, we also provide a detailed description of the Equifax-inspired environment below. The remaining details and specifications on the environments can be found in our open-source repository [67], [66].

Equifax-inspired environment. The Equifax-inspired environment has two web servers running a vulnerable version of Apache Struts with CVE-2017-5638, the same as the real environment [43]. During the Equifax breach, the attacker discovered a plaintext file on one of the web servers that included credentials to 48 different database hosts on a separate network [43].¹⁶

To replicate the databases in our environment, we create a second network with 48 database hosts and add files with fake critical consumer data such as emails, social security numbers, and addresses. On a random web server, we add a plain-text SSH configuration file that contains credentials to all the databases.

Appendix C. Token usage

We break down the token usage of Incalmo in Tab. 3. Incalmo used between 3.5K-5897.1K input tokens and 0.2K-60.1K output tokens for the planning LLMs. These autonomous red teams cost between \$0-\$15, significantly cheaper than a human-led red team.

16. From public information, it is unclear how many additional non-database credentials were in the file, but we assume that the credential file only contained database credentials.

TABLE 4. OVERVIEW OF ENVIRONMENTS IN MHBENCH

<i>Environment</i>	<i>Description</i>	<i>Goal</i>	<i>Hosts</i>
Equifax-inspired	A replica of Equifax network (same topology, services, and vulnerabilities) based on public report of the breach [43].	Exfiltrate data from 48 databases.	50
Enterprise A	A tree topology, sometimes used in enterprise networks [30], [12], with three networks. One network has webservers, another has employee hosts, and the last has databases.	Exfiltrate data from 10 databases.	30
Enterprise B	A similar topology as Enterprise A but has four networks. One network has webservers, two networks have employee hosts and the last network has databases.	Exfiltrate data from 9 databases.	40
Enterprise C	An environment inspired by the Colonial Pipeline breach [33] and other ICS attacks [38], [68]. The environment models three IT networks. One of the networks manages critical actuators with a management host.	Gain access to 15 critical actuators.	45
4-Layer chain	Each host has credentials to another host in the network [36], [72]. Each host has critical data.	Exfiltrate data from 25 databases.	25
6-Layer chain	Same topology and goal as 4-layer chain, but the data on each host requires privileged access. Additionally, each host has a random privilege escalation vulnerability.	Exfiltrate data from 25 databases.	25
4-Layer star	A single network where all hosts have a variety of remote code execution vulnerabilities. Each host has critical data. [17].	Exfiltrate data from 25 databases.	25
6-Layer star	Same topology and goal as 4-layer star, but the data on each host requires privileged access. Each host has a random privilege escalation vulnerability.	Exfiltrate data from 25 databases.	25
Dumbbell A	The topology contains two networks, one with external webservers and another with databases [40]. Each web server has credentials to a unique database.	Exfiltrate data from 15 databases.	30
Dumbbell B	Has the same topology as Dumbbell A. Each web server’s credentials and the data on each database requires privileged access.	Exfiltrate data from 15 databases.	30

Appendix D. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

The paper presents a benchmark for evaluating the ability of systems to conduct multi-host attacks across 40 emulated scenarios. It then presents a system, Incalmo, using LLM agents that can acquire critical assets in 37 / 40 of the scenarios. In contrast, naive LLM prompting and prior systems successfully obtain assets in 3 / 40 scenarios.

D.2. Scientific Contributions

- Provides a New Data Set For Public Use.
- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.
- Establishes a New Research Direction.

D.3. Reasons for Acceptance

- 1) The creation of a new dataset for multi-host attack evaluation can be valuable for future work in the space
- 2) The paper’s new system, Incalmo, outperforms prior work and naive LLM prompting, suggesting progress in the development of new security systems

D.4. Noteworthy Concerns

- 1) The paper’s 40 scenarios seem to reuse many of the same vulnerabilities/exploits, and these vulnerabilities appear to be some of the most well-documented CVEs. This homogeneity limits the generalizability of the benchmark because the attack patterns will follow very similar steps as a result even if the network topology varies between scenarios.
- 2) It is not clear from the evaluation whether the system can conduct multi-host attacks that do not follow common attack sequences discussed in public and potentially memorized reports. The paper’s main system contribution appears to be significant engineering efforts to make it so that LLMs primarily need to restate high-level attack plans that exist in the literature.
- 3) It might not be appropriate to describe Incalmo as fully autonomous. For example, the paper decides to use some pre-existing attack tools wholesale for certain tactics, such as C2, and LLM-based agents for other tactics where existing tools also exist.