

SYSEVOLVE: An AI-native, safe, autonomous adversarial attack-defense co-evolutionary system

Yuhan Meng
School of Computer
Science, Peking
University
mengyuhan@pku.edu.cn

Shaofei Li
School of Computer
Science, Peking
University
lishaofei@pku.edu.cn

Jionghao Huang
Southeast University
213230515@seu.edu.cn

Jiandong Jin
School of Computer
Science, Peking
University
jjd@pku.edu.cn

Puyi Wang
School of Computer
Science, Peking
University
wangpuyi@stu.pku.edu.cn

Hanlin Jiang
School of Computer
Science, Peking
University
jianghanlin@stu.pku.edu.cn

Anis Yusof
School of Computing,
National University of
Singapore
anis@nus.edu.sg

Peng Jiang
Southeast University
pengjiang@seu.edu.cn

Zhenkai Liang
School of Computing, National
University of Singapore
liangzk@nus.edu.sg

Yao Guo
School of Computer Science,
Peking University
yaoguo@pku.edu.cn

Ding Li¹
School of Computer Science,
Peking University
ding_li@pku.edu.cn

August 2026

Abstract

The rapid advancement of large language models (LLMs) has created a growing asymmetry in cybersecurity, where attack accelerates toward autonomous execution while defense remains predominantly human-intensive. Despite substantial prior work across cyber ranges, AI-driven attack, and AI-driven defense, this asymmetry persists. We trace it to a deeper root cause, that evolution itself has stalled on both sides at three layers. *Deployment* lacks reliable range infrastructure and trustworthy assessment. *Capability* lacks efficient attack-plan generation and adaptive detection. *Adversarial* lacks a closed-loop data flow connecting attack and defense. To overcome this, we propose **co-evolution** as the integrating insight, where attack and defense AI agents autonomously and safely drive each other’s evolution through adversarial confrontation. Based on this insight, we present SYSEVOLVE, comprising three co-designed components, SYSFIELD, SYSSPEAR, and SYSARMOR. SYSFIELD constructs realistic multi-host ranges via declarative topology-driven construction, provides complete attack-trace collection and trustworthy assessment. SYSSPEAR generates efficient, safe attack schemes via task-sliced local search, skill-based expertise enhancement mechanism, and static-analysis verification. SYSARMOR performs real-time, interpretable defense via state tracking, CTI-context fusion, and stage-matched response. Together they form a self-driven adversarial loop restoring evolution at all three layers. In evaluation, SYSFIELD achieves zero-loss collection at 2.1% overhead and orchestrates 257 CVEs into 1,148 ranges, SYSSPEAR improves attack success by over 25% over baseline LLMs, and SYSARMOR achieves 10–1000× greater precision than prior systems and detects real APT attacks in production at Huawei and Sangfor. Our evaluation also reveals three findings about LLM agent capabilities. First, multi-step composition and

¹Corresponding author.

larger topologies expose agent capability gaps hidden by single-step evaluations. Second, the bottleneck lies after initial access in post-compromise state utilization. Third, LLM agents are susceptible to environmental interference. When decoy endpoints are deployed in the range, agent timeouts triple and downstream completion disappears despite the success rates of initial accesses are unchanged.

1 Introduction

The rapid advancement of large language models (LLMs) is fundamentally reshaping the cybersecurity landscape. On the offensive side, AI agents have transitioned from answering security questions to autonomously executing multi-stage attacks. Recent systems demonstrate that LLM-based agents can autonomously exploit zero-day vulnerabilities [11, 59], conduct penetration testing at scale [9, 15], and even breach real-world protected systems [24, 42]. On the defensive side, however, security operations remain predominantly human-intensive. Security operations center (SOC) teams rely on manual alert triage, hand-written detection rules, and periodic attack-defense exercises whose outcomes rarely consolidate into lasting capability gains. This growing asymmetry—attack entering autonomous self-evolution while defense remains static—has been recognized as a critical concern by recent surveys and roadmaps [17, 37, 51].

Prior work has made substantial efforts across cyber ranges, AI-driven attack, and AI-driven defense, which are the three fronts of LLM-enhanced cybersecurity, yet the asymmetry between attack and defense persists. **Cyber ranges and evaluation platforms** such as PenGym [36], ExploitGym [49] and the CAGE [25] provide training environments for autonomous agents, but they are largely designed for reinforcement learning with fixed action spaces, support only single-point or single-CVE scenarios, and lack infrastructure to facilitate LLM-based multi-stage attack campaigns. **AI-driven attack** systems, including PentestGPT [9], Getting pwn'd by AI [15], PentestAgent [45], and autonomous zero-day exploiters [59], advance offensive capability but operate single-sidedly. Moreover, all of them require human-in-the-loop execution or target static, undefended environments, with no defensive counterpart and no mechanism for the attacker to adapt to a learning defender. **AI-driven defense** systems [5, 29, 32, 33, 44] improve detection and triage but are trained on historical data, operate reactively, and do not adapt to an active, evolving attacker. A recent measurement study further reveals that only 0.01% of SOC alerts correspond to true attacks [53], underscoring how static, non-adaptive defense fails to keep pace with evolving threats. Despite their individual advances, all three components evolve independently, with no mechanism connecting attack with defense.

This independence is not merely a coordination failure. It reflects a deeper root cause. *Evolution itself has stalled on both sides, at three layers.* First, at the **deployment layer**, neither side has access to a reliable, reusable range infrastructure that can construct and sustain realistic multi-host scenarios. Nor is there trustworthy quantitative assessment of capability. Platform failures and infrastructure limitations contaminate assessment scores. Agents fail due to infrastructure problems, not insufficient capability. Second, at the **capability layer**, the attack side relies on exhaustive autonomous exploration without efficient strategy generation, external knowledge guidance, or correctness guarantees for generated attack plans. The defense side struggles with highly imbalanced data distributions. It lacks alert interpretability due to the semantic gap between system events and threat intelligence. It cannot respond accurately and timely in dynamic environments. Third, at the **adversarial layer**, attack and defense teams train independently. Exercise outcomes do not feed the other side's capability. An automated closed-loop data flow, where attack exercises automatically drive defense model evolution and defense feedback automatically drives attack scenario upgrades, does not yet exist. Simply filling one side's missing capability, known as generation-matching, leaves the defender always a half-beat slow. The real fix must restore evolution at all three layers simultaneously.

Therefore, we propose **co-evolution** as the integrating insight. In a co-evolutionary loop, the attack team auto-generates diverse attack schemes that feed the defense team's detection model training. The defense team's detection feedback, including which attacks are caught, which slip through, and which fail, guides the attack team to refine and escalate its attack strategies. This closed-loop mutual driving restores evolution at all three layers. The range platform provides the scalable deployment substrate which is more like the real scenario. Each side's capability mechanisms en-

able self-evolution, because the adversarial loop itself provides the cross-side pressure that current generation-matching cannot.

Based on this insight, we present **SYSFIELD**, **SYSSPEAR**, and **SYSARMOR**, three co-designed components forming a self-driven adversarial co-evolution loop. **SYSFIELD** constructs and sustains realistic multi-host cyber ranges and provides trustworthy capability assessment, restoring deployment evolution. **SYSSPEAR** auto-generates attack schemes driven by external expert knowledge with static-analysis-verified correctness and safety, restoring attack capability evolution. **SYSARMOR** performs real-time threat detection, interpretable alert analysis, and timely accurate response with continual model self-evolution, restoring defense capability evolution. Together, the three components realize the adversarial co-evolution loop.

Realizing each component under the co-evolution surfaces distinct engineering challenges, each met by a targeted solution idea in our design. For **SYSFIELD**, the challenges are: (1) complete attack-trace collection under IO-intensive scenarios, addressed by an isolation-and-offloading mechanism that achieves zero-dropout collection at acceptable overhead; (2) range construction with complex multi-node topologies under limited resources, addressed by constraint-solving for deployment and state-tracing for runtime recovery; and (3) trustworthy quantitative assessment, addressed by fine-grained, process-anchored verification that prevents outcome-only scoring from being gamed. For **SYSSPEAR**, the challenges are: (1) efficient strategy generation over a large search space, addressed by task slicing and context-injected local search; (2) lack of external knowledge guidance, addressed by a codified expertise library of over 200 skills distilled from real business scenarios, organized via a skill dependency graph; and (3) attack plan correctness and safety, addressed by static validation of commands, parameters, and preconditions, plus boundary enforcement to prevent out-of-scope actions. For **SYSARMOR**, the challenges are: (1) highly imbalanced data distributions requiring real-time detection, addressed by real-time state tracking; (2) alert interpretability despite the semantic gap between system events and threat intelligence, addressed by context-aware fusion with CTI reports; and (3) response accuracy and timeliness in long-horizon, dynamically evolving incidents, addressed by state tracing and stage matching to prevent state loss and stage mismatch.

We evaluate SYSEVOLVE on real-world multi-stage attack scenarios across all three components. **SYSFIELD** achieves zero-dropout log collection at 2.1% system overhead (versus 60–98% dropout and 23–33% overhead for existing tools), orchestrates over 257 CVEs into 1,148 distinct ranges, and zero commercial LLMs escaped the range or bypassed its verification mechanism. **SYSSPEAR** improves attack success rate by over 25% compared to baseline commercial LLMs, improves inference efficiency by over 20% under local-memory and slicing mechanisms, and achieves the highest vulnerability detection rate while maintaining highest efficiency—overcoming the secure-but-time-consuming trade-off of traditional static detection. **SYSARMOR** achieves 10–1000× greater precision than **HOLMES** and **UNICORN**, detects 6 and 5 real APT attacks in Huawei and Sangfor production deployments respectively, and performs real-time detection at 21× the speed of **PROVDETECTOR** and 7× that of **UNICORN**. Using **SYSFIELD** as the evaluation substrate, we further assess current commercial LLM agents and reveal three limitations on their attack capability. First, multi-step composition and larger topologies expose capability gaps hidden by single-step evaluations. Second, the bottleneck lies after initial access in post-compromise state utilization. Third, LLM agents are also susceptible to environmental interference, as decoy endpoints triple timeouts and suppress downstream completion despite unchanged initial access rates.

This paper makes the following contributions:

- **Co-evolution paradigm.** We identify three layers of evolution lack, which are deployment, capability, and adversarial, and propose co-evolution as the integrating insight that restores evolution at all three layers, moving beyond single-sided capability enhancement.
- **SYSEVOLVE system.** We present three co-designed components, including **SYSFIELD** (range platform), **SYSSPEAR** (auto-attack system), and **SYSARMOR** (auto-defense system), that form a self-driven adversarial co-evolution loop, the first system to realize autonomous, safe, attack-defense co-evolution.
- **Component-level novel techniques.** For **SYSFIELD**, we solve complete attack-trace collection, scalable multi-host range construction, and trustworthy assessment via isolation-and-offloading, declarative topology-driven construction, and fine-grained verification. For

SYSSPEAR, we solve efficient and safe attack-plan generation via task-sliced local search, skill-based expertise enhancement mechanism, and static-analysis-verified correctness. For SYSARMOR, we solve real-time, interpretable, and timely defense via state tracking, CTI-context fusion, and stage-matched response.

- **Comprehensive evaluation on SYSEVOLVE.** conducted thorough experiments on SYSEVOLVE. Our experiments show that SYSFIELD achieves zero-loss event collection at 2.1% overhead versus 60–98% event loss and 23–33% overhead in existing tools, and can automatically orchestrates 257 CVEs into 1,148 ranges. SYSSPEAR improves attack success by over 25% over baseline LLMs. SYSARMOR detects real APT attacks in production at Huawei and Sangfor with $10\text{--}1000\times$ precision over HOLMES and UNICORN.
- **LLM agent capability findings.** Our evaluation reveals three findings about LLM agent capabilities. First, the low attack success rate by LLM agents on multi-step compositions and larger topologies exposes their limited capability hidden by single-step or small-range evaluations. Second, the bottleneck of their poor performance lies after initial access, in post-compromise state utilization rather than at entry. Third, LLM agents are susceptible to environmental interference. When decoy endpoints are deployed in the range, agent timeouts triple and downstream completion disappears despite the success rates of initial accesses are unchanged.

The rest of this paper is organized as follows. Section 2 presents background and related work. Section 3 presents the overview of SYSEVOLVE with its co-evolution loop. Section 4 details SYSFIELD. Section 5 details SYSSPEAR. Section 6 details SYSARMOR. Section 7 presents the evaluation. Section 8 concludes.

2 Background & Related Work

2.1 Cyber Attack Workflows and Measurement Boundaries

Interactive cybersecurity tasks draw from a workflow that can include reconnaissance, vulnerability discovery, exploit construction, initial access, privilege escalation, credential recovery, tunneling, lateral movement, persistence, objective completion, and reporting [30, 47]. A benchmark rarely delegates this entire chain to the agent. It instead chooses a starting point, reveals a particular amount of information, and defines an endpoint that can be checked reliably. These choices determine the capability claim supported by a score. A differentially verified proof of concept establishes vulnerability reproduction; unauthorized code execution establishes exploitation; a flag on the third host establishes multi-stage progression. None of these endpoints is a general substitute for the others.

The breadth of the delegated workflow must also be distinguished from how long an agent runs. UK AISI’s public long-horizon evaluation measures progression through a 32-step corporate-network task and a 7-step industrial-control-system task, while METR’s time-horizon metric estimates the amount of human-expert task time at a fixed success probability [12, 35]. Both are useful, but neither is identical to the number of vulnerability-research or ATT&CK stages traversed by one execution. Task starting point, stage span, interaction budget, and endpoint should therefore be reported as separate dimensions.

2.2 Existing Cybersecurity Agent Benchmarks

Cybersecurity-agent benchmarks can be organized into four broad research lines. The lines are complementary rather than a difficulty ranking: each delegates a different portion of the workflow to the agent and uses a different endpoint as evidence. Table 1 provides a family-level overview. Rather than reproduce every comparison dimension in the main text, we retain two focused tables: Table 2 isolates the task properties central to multi-host, multi-stage execution, and Table 3 isolates the controls needed for credible system-level evaluation. The remaining distinctions are synthesized in prose where they become relevant.

Within the first line, CyberSecEval spans knowledge and hazardous-capability items, while Cybench and the InterCode CTF environment add executable feedback and flag-based completion [4, 52, 55].

Table 1: Representative cyber-agent benchmark families. The endpoint states what a successful result establishes and what it does not establish by itself.

Research line	Representative evaluations	Typical task starting point	Primary endpoint and residual blind spot
Knowledge and executable CTFs	CyberSecEval; Cyberbench; InterCode-CTF [4, 52, 55]	Question, challenge statement, attachment, or interactive service	Correct item or captured flag; success need not demonstrate operation across independent enterprise hosts.
Vulnerability discovery and reproduction	CyberGym; SEC-Bench Pro; VulnLMP [27, 40, 50]	Vulnerability description and code, historical engine source, or hardened research target	Differentially or cross-version verified PoC, crash, or reproducible evidence; full exploitation and lateral movement are not always required.
Exploit construction	ExploitBench; ExploitGym [28, 49]	Known vulnerability materials or an input that already triggers the flaw	Milestone signals or independently confirmed unauthorized code execution; evaluation is typically centered on one vulnerable target.
Remote and long-horizon cyber operations	CVE-Bench; AgentCyberRange; OpenAI Cyber Range; UK AISI [12, 30, 38, 58]	Remote application, exposed service, or high-level network objective	Externally checked target state, final scenario objective, or deepest reached step; defensive observability is generally not a co-equal outcome.

The second line moves into real code. CyberGym contains 1,507 historical vulnerabilities from 188 open-source projects and accepts a generated PoC only when it triggers the pre-patch version but not the post-patch version. SEC-Bench Pro withholds original proofs of concept and patches so that agents must rediscover JavaScript-engine vulnerabilities and survive cross-version scoring [27, 50]. VulnLMP expands the research process over multiple days and parallel directions, but a long runtime does not imply that every trajectory completes the same exploitation stages [40].

ExploitBench and ExploitGym focus more directly on exploitation. ExploitBench decomposes progress into 16 verifiable milestones, from vulnerable-code coverage and crash triggering through arbitrary read/write, program-counter control, and code execution. ExploitGym begins from a proof of vulnerability that triggers the flaw and uses dynamic flags plus an independent judge to verify unauthorized code execution in user-space programs, V8, or the Linux kernel [28, 49]. These finer-grained endpoints reveal progress that a binary success label would hide, but they do not by themselves test the transfer of access, credentials, or network position across multiple hosts.

Cyber ranges observe a different segment of the workflow. CVE-Bench remotely probes real web applications without source-code access and verifies target state externally. OpenAI’s internal cyber ranges require vulnerability, configuration, credential, and lateral-movement steps to be chained within simulated networks. AgentCyberRange separates web exploitation from post-exploitation across real applications and enterprise-like multi-host ranges and supplies a common execution and verification pipeline. UK AISI’s long-horizon tasks report how deeply an agent advances through extended network objectives [12, 30, 38, 58]. Such evaluations move closer to operational intrusion workflows, but they need not measure the same depth of novel vulnerability discovery or exploit construction. The distinction is structural: vulnerability-research benchmarks go deeper into discovering and weaponizing a flaw, whereas network ranges go farther after initial access.

2.2.1 Focused Comparison of Executable Attack-Chain Coverage

The focused comparison uses three status values. *Full* means that at least one publicly documented standard configuration explicitly requires the capability within one task, or that the SYSFIELD experiment implements it with verifiable records. *Partial* means that the capability appears only in a subset of tasks, configurations, or stages, while a dash indicates that it is not a primary target. These entries describe task orientation, not an aggregate quality ranking [4, 12, 27, 28, 38, 40, 49, 50, 52, 55, 58].

Table 2: Task coverage: tool interaction and attack-chain continuity.

Evaluation	Tool Interaction	Multi-Host Attack Chain	Multi-Stage Attack Chain
CyberSecEval Series [4]	Partial	—	Partial
Cybench / InterCode-CTF [52, 55]	Full	—	Partial
CyberGym [50]	Full	—	Partial
ExploitBench [28]	Full	—	Full
ExploitGym [49]	Full	—	Full
CVE-Bench [58]	Full	—	Full
VulnLMP [40]	Full	—	Partial
SEC-Bench Pro [27]	Full	—	Full
OpenAI Internal Cyber Range [38]	Full	Full	Full
UK AISI Long-Horizon Cyber Range [12]	Full	Full	Full
SYSFIELD	Full	Full	Full

Table 2 should not be read as a scalar ranking. Vulnerability-research benchmarks may examine exploit primitives and code execution deeply without measuring cross-host operations, whereas cyber ranges may observe continuous network actions without requiring novel vulnerability discovery. A *multi-host* chain transfers privileges, credentials, or access paths across independent hosts or services; a *multi-stage* chain consecutively completes multiple security phases. Extended runtime, more interaction rounds, or a dataset containing multiple task categories does not by itself establish either property.

Task starting points and success endpoints further bound the resulting claims even though they are not expanded into another table. CyberGym and SEC-Bench Pro begin from known vulnerability materials and terminate at a verified PoC; ExploitBench and ExploitGym continue toward exploit primitives or unauthorized execution; cyber ranges instead emphasize initial access and subsequent network operations. SYSFIELD evaluates known-CVE exploitation and continuous progression through credential or privilege acquisition, lateral movement, and externally verified final objectives, rather than novel vulnerability discovery. METR’s time-horizon metric remains useful for duration, but it cannot substitute for vulnerability-research or ATT&CK stage coverage [12, 27, 28, 35, 38, 47, 49, 50, 58].

2.3 Existing Evaluation Practice for Interactive Agents

An agent harness translates model outputs into system actions. It chooses the command interface, retains observations, manages context and state, handles timeouts and malformed tool calls, retries failures, and decides when a task has terminated. These choices can expose or suppress capabilities of the same backbone model. Cybench compares structured commands, action-only interaction, pseudo-terminals, and optional web search; CyberGym reports substantial differences among software agents operating over the same vulnerability corpus; ExploitBench separates unified runners, dynamically prompted systems, and vendor-native command-line agents; and AgentCyberRange uses adapters to expose heterogeneous CLI agents through a common interface [28, 30, 50, 55]. Thus, “model performance” is underspecified unless the harness, tools, permissions, knowledge, budgets, and termination policy are reported with the model.

The execution environment is an additional experimental factor. A disclosed incident around an internal ExploitGym-related evaluation involved exploitation of a package-caching component and showed how egress, credentials, isolation, and monitoring can become both measurement variables and security boundaries [18, 41]. Range qualification, reset behavior, service availability, and infrastructure exclusions therefore belong in the evaluation record. Without them, reasoning failure cannot be separated from a missing executable, lost state, a terminated process, an unavailable service, or an exhausted budget.

Verification practice has advanced alongside task realism. CyberGym’s pre-/post-patch oracle, ExploitBench’s capability signals, ExploitGym’s dynamic flags and independent judge, CVE-Bench’s

state verifier, AgentCyberRange’s runtime verification, and UK AISI’s deepest-reached-step reporting all reduce dependence on model self-report [12, 28, 30, 49, 50, 58]. Process traces and milestone labels further distinguish partial progress from terminal success. Public reports nevertheless vary in their treatment of repeated trials, infrastructure failures, cost, system controls, and failure attribution.

For the present work, one boundary remains especially important. An offensive oracle can establish that a trigger fired, a flag appeared, or a target state was reached; it does not establish whether a runtime defense observed the behavior. Conversely, a security signal can establish that behavior matched an observation rule; it does not establish successful compromise. Existing evaluations provide important pieces of a rigorous methodology, but they do not commonly treat offensive progression and defensive observability as co-equal, independently verified outcomes of the same trajectory.

2.3.1 Focused Comparison of Evaluation Controls

Broad task coverage does not by itself make a score interpretable. Table 3 focuses on the three controls most directly required by SYSFIELD: environment qualification, isolation of system factors, and external outcome verification. It applies the same *Full*, *Partial*, and *Not disclosed* convention used above [4, 12, 27, 28, 38, 40, 49, 50, 52, 55, 58].

Table 3: Evaluation methodology: environment, factor isolation, and verification.

Evaluation	Environment Qualification	Isolation of System Factors	External Outcome Verification
CyberSecEval Series [4]	Partial	Partial	Partial
Cybench [55]	Full	Full	Full
InterCode-CTF [52]	Partial	Partial	Full
CyberGym [50]	Full	Partial	Full
ExploitBench [28]	Full	Full	Full
ExploitGym [49]	Full	Partial	Full
CVE-Bench [58]	Partial	Full	Full
VulnLMP [40]	Not disclosed	Partial	Full
SEC-Bench Pro [27]	Full	Partial	Full
OpenAI Internal Cyber Range [38]	Not disclosed	Full	Partial
UK AISI Long-Horizon Cyber Range [12]	Not disclosed	Not disclosed	Partial
SYSFIELD	Full	Full	Full

Table 3 exposes three distinct threats to interpretation. Without environment qualification, infrastructure failure can enter the outcome denominator; without factor isolation, a model score may conflate the backbone with its harness, tools, or execution policy; and without external verification, terminal success may depend on agent self-report. These controls are complementary rather than interchangeable.

Dimensions omitted from the table remain reporting requirements. Trial structure and stochastic limitations must be stated, process records must preserve stage progression and termination, failures must be attributed where evidence permits, and runtime or monetary cost must be reported in the unit supported by the protocol. Defensive observation is kept separate from the offensive oracle: a verified compromise does not imply that the behavior was observed, and a security signal does not establish compromise.

This synthesis motivates SYSFIELD. Comparable results require the task start and endpoint, evaluated system, tool and knowledge boundary, budget, environment qualification, and verifier to be explicit. Credible multi-stage results additionally require stage-level progression, attributable failures, reproducible records, and a separate defensive oracle. The two focused comparisons retain the evidence most directly tied to these requirements without turning the background into an exhaustive benchmark catalog.

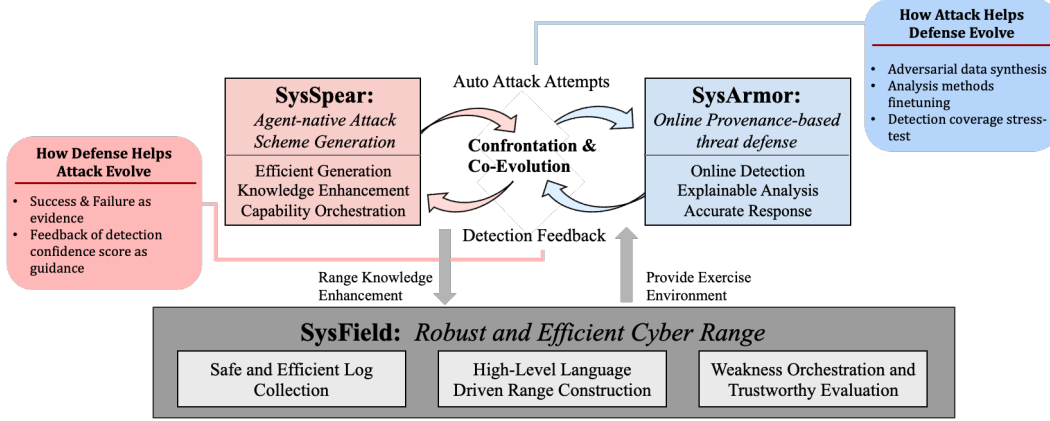


Figure 1: Overview of SYSEVOLVE. Three components form a self-driven adversarial co-evolution loop. SYSFIELD provides the range substrate, SYSSPEAR generates attack schemes, and SYSARMOR detects and responds. Attack outcomes feed defense improvement, and detection results feed attack refinement.

3 Overview of SYSEVOLVE

SYSEVOLVE is an AI-native autonomous adversarial attack-defense co-evolutionary system comprising three co-designed components. SYSFIELD constructs and sustains realistic multi-host cyber ranges and provides trustworthy capability assessment, serving as the deployment substrate for both sides. SYSSPEAR auto-generates attack schemes driven by expert knowledge and static-analysis-verified correctness, serving as the attackers. SYSARMOR performs real-time threat detection, interpretable alert analysis, and timely response with continual model self-evolution, serving as the defense. Together, the three components form a self-driven adversarial co-evolution loop in which each side’s outputs drive the other’s capability improvement.

The co-evolution loop operates in two directions. In the attack-to-defense direction, SYSSPEAR’s generated attack schemes serve as adversarial data that stress-tests SYSARMOR’s detection coverage. When SYSARMOR successfully detects an attack, the detection model is reinforced. When an attack slips through, the missed detection exposes a coverage gap that drives model finetuning. This transforms red team exercise outcomes from one-time reports into continual training signal, enabling SYSARMOR’s analysis methods to evolve against realistic, changing attack patterns rather than historical data alone.

In the defense-to-attack direction, SYSARMOR’s detection results feed back to SYSSPEAR as guidance for attack refinement. Each attack action’s success or failure becomes evidence recorded in the Strategy Board, informing whether the action’s approach is viable and worth refining. Beyond binary success or failure, SYSARMOR’s detection confidence scores provide finer-grained guidance. High confidence on a detected action indicates that the approach is easily visible to the defender, prompting SYSSPEAR to seek stealthier alternatives. Low confidence on an undetected action confirms that the approach evades current detection, encouraging SYSSPEAR to develop it further. This transforms blue team detection results from passive outcomes into active guidance for the red side.

This bidirectional feedback restores evolution at all three layers. SYSFIELD provides the deployment substrate where adversarial exercises occur and are assessed. SYSSPEAR and SYSARMOR each possess capability mechanisms that enable self-evolution. The adversarial loop itself provides the cross-side pressure, ensuring that each side improves not in isolation but in response to the other’s evolving capability.

4 SYSFIELD

4.1 Design Overview

SYSFIELD is the evaluation substrate of SYSEVOLVE, designed to make autonomous cyber-agent evaluation credible, reproducible, and process-grounded. Its design is driven by three system prob-

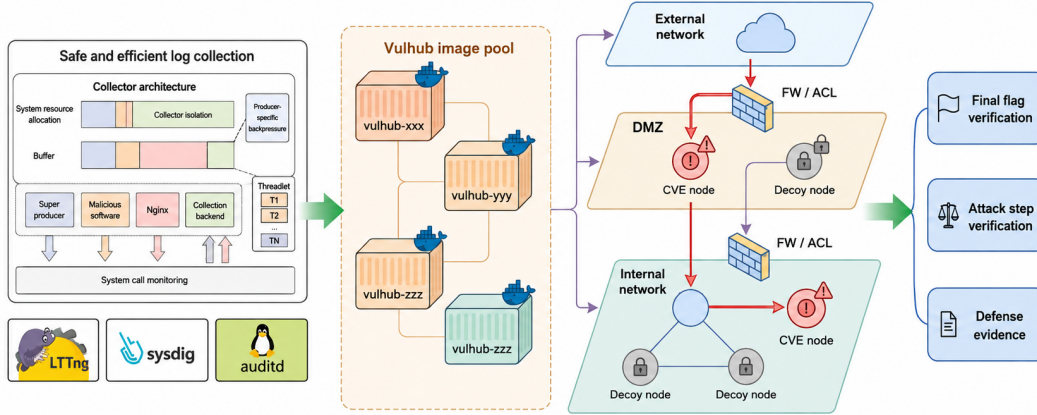


Figure 2: Overview of SYSFIELD. SYSFIELD integrates three tightly coupled planes: safe and efficient log collection, CRADLE-guided cyber-range construction from reusable vulnerable images, and context-aware weakness orchestration with multi-layer verification. Together, they provide execution evidence, reproducible range deployment, and process-grounded evaluation of multi-stage cyber attacks.

lems. First, an attack can generate audit events faster than a conventional centralized collector can consume them, causing the very evidence needed for evaluation to be dropped when the workload is most security relevant. Second, realistic enterprise ranges contain heterogeneous hosts, networks, software versions, and resource constraints that cannot be reliably constructed or maintained through a fixed collection of hand-written deployment scripts. Third, a realistic multi-stage campaign is not merely a set of independent vulnerabilities, and accepting only a final flag rewards shortcuts that bypass the intended attack process or even escape the range. Accordingly, SYSFIELD is organized around three corresponding solutions: *safe and efficient log collection*, *High-Level Declarative Language Driven Scalable Cyber Range Topology Construction*, and *context-aware weakness orchestration with trustworthy verification*.

Figure 2 illustrates how these three solutions interact in the overall SYSFIELD workflow. The collection plane captures execution evidence through isolated or offloaded logging mechanisms. CRADLE then materializes the cyber range from reusable vulnerable images and a high-level topology description, constructing segmented environments such as the DMZ and internal network. On top of this range, the weakness orchestration and verification plane instantiates attack paths, monitors ordered progress, and validates both attack success and defense-side evidence.

These solutions form a dependency-ordered control loop. The collection plane preserves the execution evidence. CRADLE constructs and maintains the topology in which an experiment runs. The weakness orchestrator binds compatible vulnerabilities to that topology and admits a case only after its intended path and containment boundary have been validated. During evaluation, the verifier consumes environment state together with the collection plane’s audit evidence, while its private objectives and control interfaces remain inaccessible to the evaluated agent. In this way, SYSFIELD cleanly separates the system being attacked from the mechanisms that determine whether the attack is valid.

Table 4: The three core design solutions in SYSFIELD.

System challenge	Core solution	Design outcome
Complete evidence competes with finite host resources and can be disrupted by event floods	Isolated NoDrop collection and DPUaudit offloaded collection	A common, gap-detectable event stream whose failure behavior is explicit under both software-only and hardware-assisted deployment
Complex topologies contain heterogeneous and conflicting deployment constraints and may fail during long-running materialization	CRADLE high-level description, constraint compilation, desired-state execution, and checkpointed recovery	A portable topology specification plus a reproducible deployment state and recovery history
Independent vulnerabilities do not automatically form executable attack chains, while result-only scoring rewards shortcuts	Context-aware weakness composition, environment/component isolation, and multi-layer multi-source verification	A contained range with an execution-confirmed attack path and process-grounded evaluation evidence

4.2 Safe and Efficient Log Collection

The first design goal is to preserve complete, security-relevant execution evidence without allowing collection to dominate the workload being measured. This is difficult because a collector has two distinct failure modes. A bounded collector may preserve application performance by dropping events, allowing an attacker to hide malicious actions behind a high-rate *super producer*. An unbounded centralized collector may instead consume additional CPU and memory until the monitored service, or the entire host, is impaired. SYSFIELD therefore does not treat a larger shared buffer as a sufficient solution. It provides two collector backends with the same output contract: NoDrop isolates the cost of collection at the producing thread, while DPUaudit moves the expensive collection path to an isolated DPU.

4.2.1 Common Capture and Stream Contract

SYSFIELD reuses native audit facilities wherever possible. On Linux, capture rules are expressed over the Linux Audit interface [43]; on Windows, the corresponding adapter uses Event Tracing for Windows [34]. The rule set covers process creation and termination, file-system operations, socket and network-connection activity, authentication, and privilege changes. This preserves the deployment advantages of existing audit infrastructure while allowing NoDrop or DPUaudit to replace the vulnerable consumption path when stronger loss and interference guarantees are required.

All backends emit a common event envelope containing the case and collector identifiers, host and process/thread identities, event type, timestamp, a collector-local monotonic sequence number, and the original audit payload. Sequence numbers and per-batch acknowledgments make gaps and re-transmissions observable; SYSFIELD never interprets the absence of an event as proof that the behavior did not occur unless the collector’s continuity record is complete. The original payload and metadata are retained in the raw stream, while normalization produces a stable JSON representation for downstream provenance construction.

Events are forwarded through a reliable router to a small, fixed set of partitioned topics. The data plane separates *Raw*, *Events*, and *Alerts*: *Raw* stores original audit records, *Events* stores normalized records, and *Alerts* stores derived security signals. Partitioning by collector identifier preserves endpoint-local causal order and supports balanced horizontal scaling. Normalization, rule-based analysis, and asynchronous model inference are independent stages connected by the topic layer, so a slow inference service does not block collection. Checkpoints, idempotent event identifiers, buffering, and retry isolate transient transport and processing failures without silently duplicating or discarding critical events.

4.2.2 NoDrop: Isolation-Based Collection

NoDrop addresses the super-producer threat by restoring the resource and data isolation that centralized audit collectors remove [22]. Instead of asking one shared consumer to process events from every application, each producing thread consumes the provenance data that it generates using its

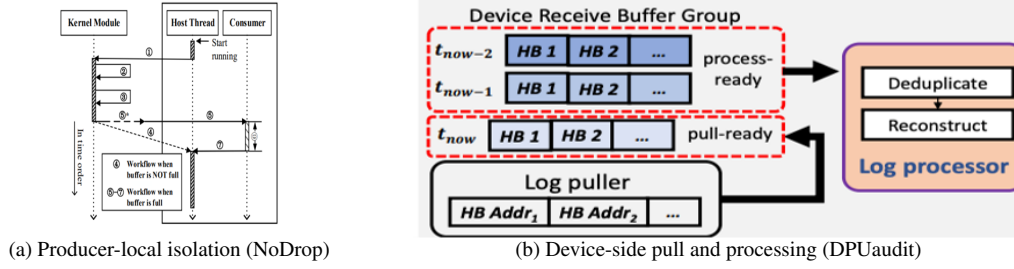


Figure 3: The two complementary loss-resilient collection paths used by SYSFIELD. NoDrop charges full-buffer draining to the producing thread to prevent cross-producer eviction [22], while DPUaudit employs a DPU-assisted pull architecture with device-side buffer rotation, record reconstruction, and deduplication [23]. The mechanism diagrams are adapted from the corresponding source papers.

own resource quota. A dedicated per-thread kernel buffer separates that thread’s events from all other producers. When the buffer fills, a lightweight consumer is inserted as a threadlet into the producing thread and drains the buffer synchronously. Consequently, an event flood can delay the producer responsible for the flood, but it cannot evict another process’s audit events or borrow unbounded resources through a shared collector.

SYSFIELD uses NoDrop when a software-only deployment is required or when the monitored node does not expose a suitable offload device. Its integration preserves three properties. First, *zero-loss backpressure*: a full buffer transfers control to the same producer’s consumer rather than overwriting an unrelated event. Second, *performance isolation*: audit processing is charged to the source thread’s scheduling and resource domain. Third, *collector protection*: the threadlet’s code, stack, heap, and mapped buffer are isolated from the host thread, preventing the application from directly modifying the consumer state. These properties address collection floods and cross-producer interference; kernel compromise and post-collection storage integrity remain separate trust assumptions and are handled by the external verifier and remote evidence store.

4.2.3 DPUaudit: Offload-Based Collection

DPUaudit is the hardware-assisted backend for dense I/O and multi-core workloads [23]. Its key principle is to leave only interception and append operations on the monitored CPU, while moving buffer draining, reconstruction, deduplication, routing, and persistence to an isolated DPU. This reduces both the host resource cost and the attack surface of a host-resident log sender. The design follows a DPU-assisted pull-based architecture: the device continuously pulls host log buffers using DMA rather than waiting for a host process to push data. Because the pulling logic is outside the monitored host, application scheduling pressure cannot starve the sender, and a compromised workload cannot directly suspend the device-side puller.

The host allocates NUMA-aware append-only buffers and publishes their descriptors to DPUaudit during trusted initialization. The device uses scatter-gather DMA to pull multiple non-contiguous buffers in one request and rotates receive buffers between *pull-ready* and *process-ready* states, allowing transfer and processing to proceed concurrently. A device-side processor uses producer indices and sequence numbers to remove overlap introduced by whole-buffer pulls, reconstruct complete events, and persist only newly committed records. The monitored host therefore performs no per-batch notification and no heavy log-processing or protection logic. DPUaudit exports the same event envelope as NoDrop, so the routing and verification layers are independent of which collection mechanism is selected.

Figure 3(b) makes the offload boundary explicit: host buffers are pulled into device-owned receive groups before reconstruction and persistence. The two backends are complementary rather than redundant. NoDrop gives process-level isolation on commodity hosts; DPUaudit gives physical and scheduling isolation when an offload device is available. A run manifest records the selected backend, buffer configuration, device/driver version, continuity counters, and any degraded mode. This makes collection behavior part of the experiment provenance and prevents an apparent agent failure or missing defensive signal from being confused with a collector failure.

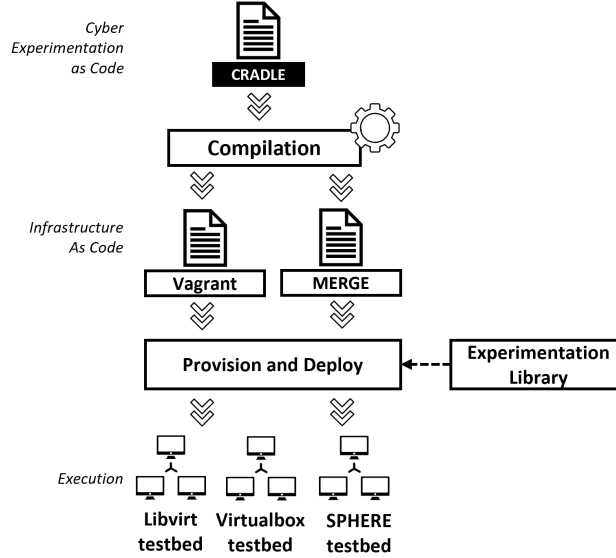


Figure 4: CRADLE is compiled into backend-specific IaC, then provisions the same logical range on heterogeneous testbeds.

4.3 High-Level Declarative Language Driven Scalable Cyber Range Topology Construction

The second design problem is to construct complex, reusable topologies under finite resources. A topology is more than a list of virtual machines: it includes operating systems and versions, software objects, network segments, addresses, routing and firewall policies, placement rules, resource budgets, and the order in which infrastructure becomes ready. These constraints are heterogeneous and may conflict. Moreover, a deployment containing dozens of nodes may fail after substantial work has already completed. SYSFIELD uses CRADLE as the management and guidance layer that translates a high-level range description into a constraint-checked, stateful deployment.

4.3.1 Range Description as Code

We design a novel high-level declarative language to construct the low-lever cyber ranges. CRADLE exposes a concise, declarative language with two linked sections. The *environment* section describes networks, endpoints, instances, operating systems, software/components, and configurations. The *events* section describes ordered setup and experiment actions together with their timing and dependencies. Separating these sections keeps the stable topology independent from a particular attack campaign while still making the actions that initialize and exercise the range reproducible. Reusable operating-system images, application packages, monitoring components, and configuration fragments are resolved through a versioned component repository.

The **Cyber-testbed Reconstruction and Automation Description Language** (CRADLE) [1] is testbed-agnostic. CRADLE compiles the high-level description into the Infrastructure as Code representation (IaC) supported by the selected testbed provider rather than exposing testbed-specific details to the scenario author. During compilation, it resolves image and component versions, assigns addresses and network segments, checks firewall reachability, and verifies resource and placement constraints. The compiler emits both provider artifacts and a normalized topology manifest, so equivalent logical ranges can be reproduced across supported backends.

4.3.2 Desired-State Execution and Recovery

CRADLE does not stop after generating deployment scripts. As summarized in Figure 4, compilation decouples one logical range description from the provider used to execute it. Its control plane consists of a System Executor, provider adapters, a State Backend, and a Checkpoint Store. The topology manifest defines the desired state; provider observations define the current state. The System Executor advances through explicit phases—resource preparation, network construction, in-

stance launch, component configuration, readiness checks, and experiment activation—and records every state transition. Operations are designed to be idempotent, allowing a partially completed phase to be retried without duplicating already materialized resources.

CRADLE creates a checkpoint after each stable phase and before every destructive transition. When a node or provider operation fails, it compares desired and observed state to localize the failed component and its dependents. A recoverable local failure triggers repair or bounded retry; a violated constraint or corrupted dependency causes rollback to the latest consistent checkpoint. Independent branches can continue while the affected subgraph is repaired. This state model turns interruption handling into a first-class orchestration property: the output of topology construction includes not only the deployed nodes, but also the exact manifest, provider bindings, readiness evidence, checkpoints, and recovery history that produced the running range.

The resulting topology deliberately contains typed *weakness slots* rather than hard-coding CVEs into the infrastructure language. CRADLE is responsible for where a weakness can be placed and which environmental constraints it must obey; the weakness orchestrator is responsible for selecting vulnerabilities whose requirements and post-compromise capabilities form a valid attack path. This boundary keeps topology management reusable while preventing deployment details from being conflated with attack semantics.

4.4 Context-Aware Weakness Orchestration and Trustworthy Evaluation

A realistic attack campaign rarely consists of a single vulnerability. Later attack stages often depend on capabilities obtained earlier, such as command execution, internal network access, or valid credentials. Simply deploying multiple vulnerable services in the same topology therefore does not guarantee that they form an executable attack path. Meanwhile, once autonomous attack agents are given access to an executable range, the evaluation infrastructure itself becomes part of the attack surface. If evaluation relies only on a terminal flag or target state, an agent may obtain reward by escaping the range, attacking the verifier, or bypassing the intended attack process rather than demonstrating the capability that the benchmark intends to measure.

SYSFIELD addresses these two problems jointly. Context-aware weakness orchestration constructs multi-weakness ranges whose capability and runtime dependencies are explicitly satisfied and validated through execution. A trusted evaluation boundary and multi-layer verification further ensure that successful results are attributed to legitimate attack progression rather than unintended shortcuts or *Hack Rewarding*.

4.4.1 Weakness Capability Atomization

As illustrated in Figure 5, SYSFIELD first normalizes each reproducible weakness into a reusable *Weakness Capability Atom*. In the current CVE-backed implementation, the Atom follows the capability interface introduced by RangeFactory [21]. Each Atom binds four types of information: (1) structured exploit preconditions (*exploit access*); (2) post-compromise capabilities confirmed by environment-side probes (*capability grants*); (3) an execution-confirmed exploitation procedure together with its runtime requirements (*exploit guide*); and (4) referenced execution assets such as PoCs and scripts (*poc materials*). Only capabilities confirmed through actual execution are admitted to the orchestration interface.

Atomization separates *what* a weakness requires and provides from *how* a particular exploit is executed. This allows heterogeneous weaknesses to share a common capability vocabulary, such as command execution, file access, internal network vantage, credential acquisition, and authenticated access. From validated executions, SYSFIELD extracts two forms of dependency information: *capability dependencies*, which characterize capability transfer across weaknesses, and *runtime dependencies*, which capture the environmental conditions required by concrete attack procedures. These dependencies provide the basis for composing independently validated weaknesses into larger attack scenarios.

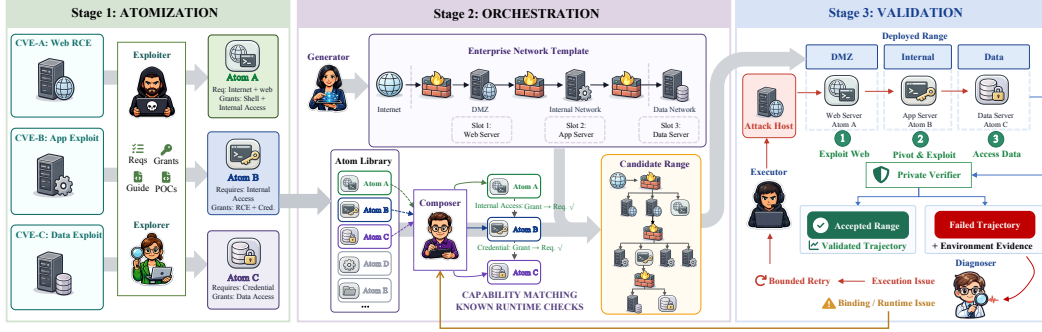


Figure 5: Overview of context-aware weakness orchestration in SYSFIELD. Reproducible weaknesses are first normalized into capability-oriented Atoms. Dependency-compatible Atoms are then composed into enterprise-range templates, and the resulting candidate ranges are validated through end-to-end execution with a private verifier.

4.4.2 Dependency-Guided Weakness Composition

The orchestration and validation stages are shown in the middle and right parts of Figure 5. Given a topology template and an Atom library, SYSFIELD composes weaknesses according to two types of dependencies recovered from validated attacks.

Capability dependencies describe whether capabilities granted by earlier weaknesses satisfy the exploit access required by subsequent ones. For example, a foothold obtained on a perimeter host may provide the internal network vantage required to reach a later service, while recovered credentials may enable authenticated access to another target. *Runtime dependencies* capture the environmental conditions required for a concrete attack procedure to succeed, including network reachability, authentication context, exposed endpoints, service configuration, and other execution-specific requirements observed during prior successful attacks.

A partial chain is pruned as soon as either type of dependency is unsatisfied. However, dependency matching alone is not treated as proof of executability, since jointly deployed weaknesses may introduce runtime conditions that were not observed when each weakness was validated independently. SYSFIELD therefore instantiates every completed candidate and validates it through end-to-end execution. As shown in the validation stage of Figure 5, a private verifier checks the ordered objectives without exposing them to the execution agent. Transient execution failures may receive bounded retries, while persistent composition failures are recorded as incompatibility evidence for subsequent orchestration. The resulting range is therefore not merely a collection of vulnerable services, but an environment with at least one execution-confirmed multi-stage attack path.

4.4.3 Trusted Evaluation Boundary against Hack Rewarding

Executable attack environments do not automatically provide trustworthy evaluation. When an autonomous agent is optimized for a terminal flag or target state, it may obtain reward through unintended shortcuts, such as escaping the designated range, accessing external resources, attacking the verifier, or reading benchmark-private state. SYSFIELD addresses such *Hack Rewarding* behaviors by establishing a trusted evaluation boundary through environment and component isolation.

At the environment level, each experiment is confined to an explicitly bounded range network that exposes only the services and communication paths required by the task. External systems, unrelated experiments, and platform-management services remain outside this boundary. At the component level, the orchestration controller, private verifier, scoring state, and telemetry infrastructure are separated from the agent-controlled workload plane, and verification predicates and private state are never exposed through the task interface. Consequently, compromising an intended target is part of the evaluation, whereas accessing or modifying the evaluation infrastructure is recorded as a boundary violation rather than a valid success. This isolation removes infrastructure-level reward shortcuts before capability results are scored.

4.4.4 Multi-Layer Fine-Grained Verification

Isolation prevents direct attacks on the evaluator, but outcome-only scoring can still reward shortcuts that bypass the intended attack process. SYSFIELD therefore verifies each execution at three levels: *environment qualification*, *capability progression*, and *terminal objective*. Before scoring, the platform checks that required nodes and services are ready, expected network paths are reachable, and the private verifier is operational, so that infrastructure failures are separated from genuine agent failures. During execution, key capability grants derived from the weakness dependency chain are used as private milestones, allowing SYSFIELD to verify whether security capabilities such as command execution, network access, credential acquisition, and authenticated access are actually established along a valid attack path without requiring the agent to reproduce a fixed command sequence. The final flag or business objective is then independently verified from environment-side state.

SYSFIELD further correlates evidence from multiple independent sources: environment-side verifier state captures *what capability was achieved*, the agent harness and tool trace record *what the agent executed*, and out-of-band audit or provenance telemetry records *what behavior was independently observed*. A result is accepted as a trustworthy success only when the environment is qualified, no evaluation-boundary violation occurs, the terminal objective is satisfied, and the required capability progression is supported by a valid attack path. Defensive telemetry is retained as an independent observability dimension rather than a prerequisite for offensive success, allowing attack progression and defensive visibility to be measured separately from the same execution.

4.5 Design Summary

The three mechanisms form a single range lifecycle. CRADLE first materializes a constrained, traceable topology and maintains its runtime state. The weakness orchestrator then binds dependency-compatible security weaknesses to that topology and validates at least one intended end-to-end path. During agent execution, the isolated collection plane records security-relevant events with bounded host impact, while the trusted verifier evaluates both terminal objectives and intermediate progress. Finally, environment state, process milestones, and independent defensive evidence are aligned into one auditable execution record.

This organization changes the role of SYSFIELD from a passive benchmark harness to a trustworthy security-evaluation substrate. The platform does not assume that a deployed range is valid because deployment returned successfully, that a vulnerability chain is valid because its metadata matches, that logs are complete because a collector is running, or that an agent is capable because a final flag appears. Each of these claims is established by a dedicated mechanism—topology orchestration and recovery, execution-grounded weakness composition, isolated lossless collection, and multi-layer verification—which together provide the foundation for reliable attack–defense co-evolution.

5 SYSSPEAR

5.1 Design Overview

The design goal of SYSSPEAR is to autonomously generate multi-stage attack schemes that are efficient, knowledge-enhanced, and verified correct and safe, restoring attack capability evolution in the co-evolution loop. This is challenging because an LLM-based attacker must simultaneously address three difficulties. The search space of a realistic campaign is large, and most branches reveal themselves as dead ends only after execution. The agent lacks domain-specific expertise, hallucinating exploit steps and missing business-specific misconfigurations. And the generated plan may contain incorrect dependencies or unsafe software objects. Efficient generation, knowledge enhancement, and formal verification are all necessary, and none can be sacrificed for the others.

Figure 6 illustrates the structure of SYSSPEAR. At the center of SYSSPEAR’s design is the *Strategy Board*, a novel abstraction of the inference memory during the attack scheme generation process that facilitates knowledge alignment and evolution among all subsystems. Around this central abstraction, SYSSPEAR addresses the three difficulties through three co-designed subsystems. A high-efficiency attack scheme generator mediates collaboration among isolated solvers through task slicing, context injection, and runtime isolation, enabling adaptive local search over a large evidence

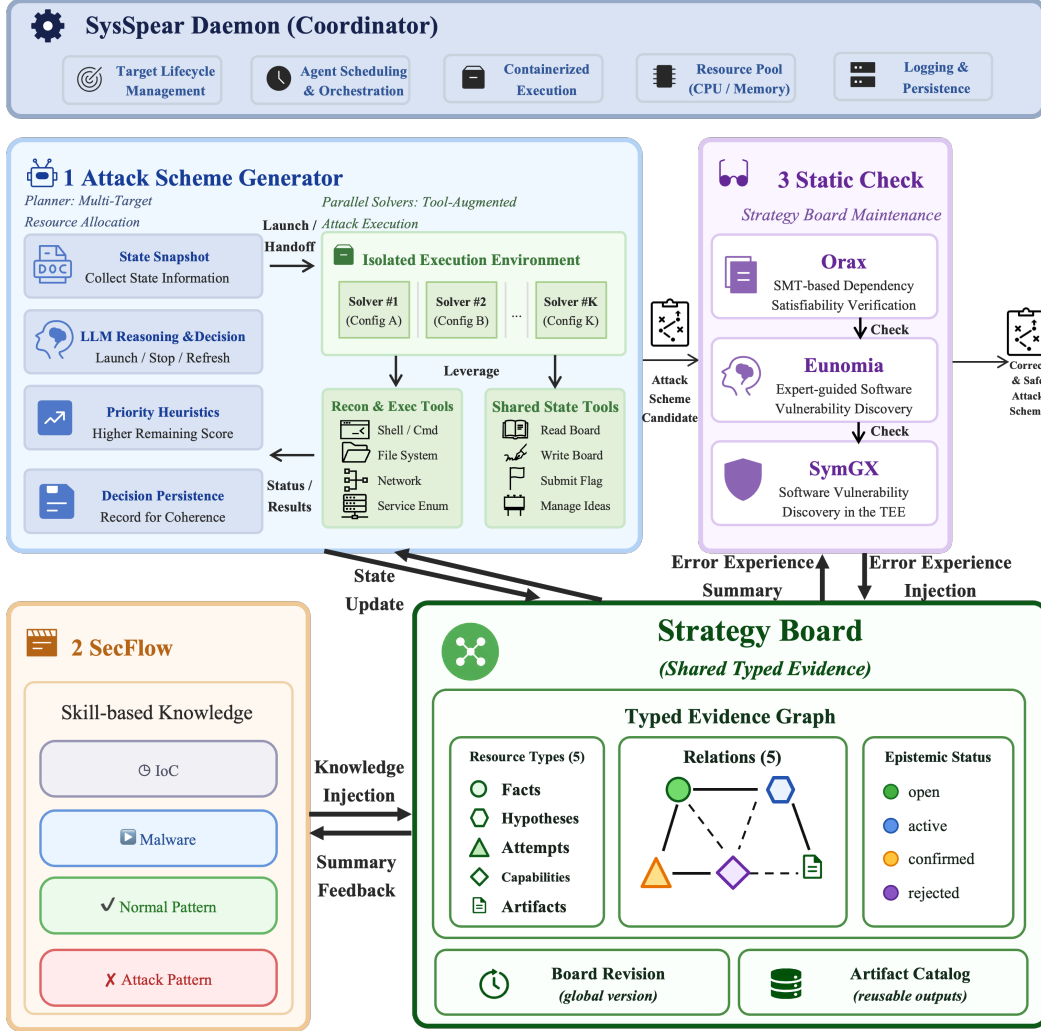


Figure 6: Overview of SYSSPEAR. Three co-designed subsystems interact around the Strategy Board for knowledge alignment and evolution. With the guidance by expert knowledge from SecFlow, the attack scheme generator produces attack scheme candidates. The Static Check component then verifies correctness and safety of them, yielding a verified attack scheme as the output of SYSSPEAR.

space. A knowledge-enhanced agent runtime, SecFlow, codifies over 200 expert skills from more than a decade of security operations into a dependency-structured library. It orchestrates heterogeneous agent runtimes through a uniform protocol and ensures reliable concurrent execution through file-system-grounded state management. Finally, a static-analysis-verified checking layer formally verifies dependency satisfiability via SMT solving and checks software safety through fine-grained symbolic execution and SGX-specific analysis. The three subsystems are not independent modules but co-designed layers that interact around the Strategy Board, ensuring that generated plans are efficient, knowledge-enhanced, and verified correct and safe before execution.

5.2 High-Efficiency Attack Scheme Generator

The first design goal of SYSSPEAR is to generate a multi-stage attack scheme efficiently. This is difficult because the effective search space of a realistic campaign is large: an attacker must compose actions across many hosts, services, and stage dependencies, and most branches reveal themselves as dead ends only after execution. A single agent that pursues this search in one conversation accumulates an increasingly noisy transcript, so its later decisions degrade precisely when the accumulated evidence matters most, while a naive multi-agent attacker merely replaces one long context with

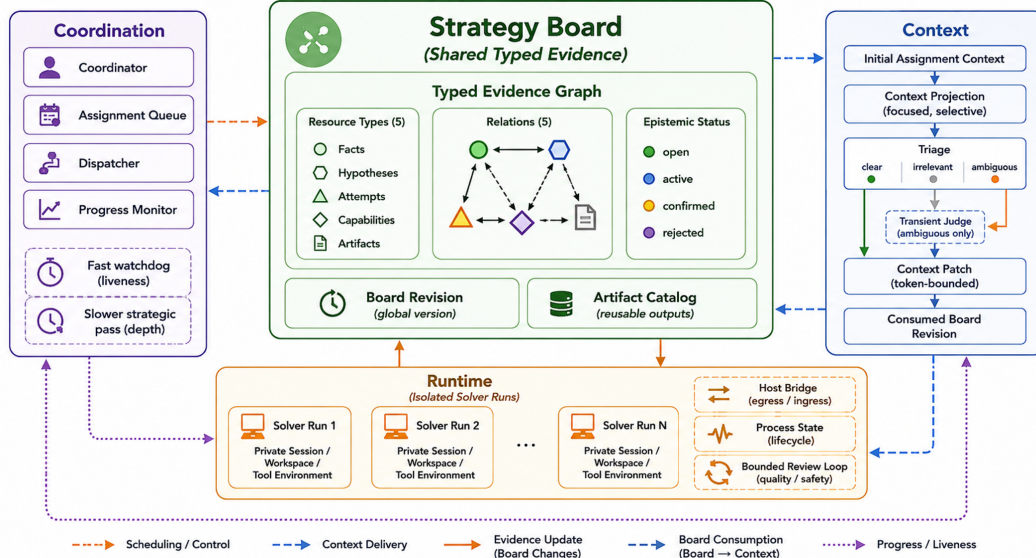


Figure 7: Strategy-Board-mediated attack-scheme generation in SYSSPEAR. The Coordinator slices the current Board into bounded Assignments; Context Projection delivers only each Assignment’s relevant subgraph; isolated Solver Runs return structured evidence to the same versioned Board.

several disconnected or redundant ones that rediscover the same facts and repeat the same failed attempts. The real problem is therefore not how to run more agents, but how to preserve useful discoveries, allocate independent lines of work, and deliver the right evidence to each executor without sharing every interaction.

SYSSPEAR addresses this with a key insight: the attack scheme is itself an evolving, typed evidence graph—the *Strategy Board*—that is at once the artifact being generated and the sole medium through which isolated executors collaborate. Unlike naive task-tree multi-agent schemes, in which each agent sees only its parent’s hand-off and returns a summary, the Board connects typed resources through explicit relations, so evidence produced anywhere is linked into the whole scheme and actively projected to the executors it affects rather than being confined to one branch. Three mechanisms realize this insight, each removing one obstacle to efficient generation: task slicing carves bounded assignments out of the Board’s current evidence, so exploration proceeds as local search that adapts to what has been learned rather than one degrading global pass; context injection is how the Board actively delivers evidence, projecting a focused sub-graph to each assignment instead of broadcasting everything or nothing; and runtime isolation keeps each executor’s session private while confining all durable collaboration to the Board, so findings reach others only as related evidence merged into the single shared graph.

5.2.1 Task Slicing into Bounded Assignments

The first obstacle is scheduling an open-ended, multi-stage search without pretending that the complete campaign is known in advance. SYSSPEAR therefore treats an *Assignment*, rather than an agent or a chat session, as the durable unit of work. Each Assignment records one concrete instruction, an independently checkable success criterion, focused Board-entry identifiers, and a reserved evidence question with its validation boundary. This makes the intended scope explicit: one Solver resolves one question using a specified evidence neighborhood, rather than a generic agent being asked to “continue the attack.”

The Coordinator forms an Assignment from a compact Board view containing confirmed facts, active hypotheses, prior attempts, available capabilities, and produced artifacts. Slicing is consequently adaptive rather than a one-time decomposition. Early Assignments can examine a broad uncertainty, whereas subsequent evidence narrows the focus to a service surface, a prerequisite, or a validation step. The resulting scheme is a succession of Board-steered local searches: each one can advance, refute, or delimit a route, and all of these outcomes are useful inputs to the next scheduling deci-

sion. Assignment *phase* (queued, running, waiting, or finished) is kept separate from its terminal *outcome*; ending a model turn therefore does not erase partial progress or falsely conclude that a route has been resolved.

Slicing also gives parallelism a precise boundary. The Coordinator may launch complementary Assignments whose focus regions differ, but a reserved evidence question prevents two Solvers from racing on the same stateful boundary. Checkpoints, produced Board entries, and a terminal outcome remain attached to the Assignment even if its Solver is replaced. Thus, concurrency expands coverage across genuinely different routes while the Board retains both successful evidence and negative results needed to suppress redundant rediscovery.

5.2.2 Context Injection of a Focused Sub-graph

Task slicing controls the question posed to a Solver; context injection controls the evidence used to answer it. The full Strategy Board is intentionally richer than any single Assignment needs, so neither an all-to-all broadcast nor an isolated hand-off is acceptable. SYSSPEAR instead constructs a token-bounded projection from the Assignment’s focus entries, instruction, and success criterion over the Board’s explicit relation graph. The result is a connected evidence neighborhood: for example, a capability is delivered together with the prerequisite facts, supporting artifact, and next route it unlocks, rather than as a disconnected summary or a full Board dump.

Projection first uses the Assignment’s focus entries, instruction, and success criterion as seeds. It then propagates relevance along explicit evidence relations, gives priority to confirmed evidence and to *contradicts* and *unlocks* edges, and retains a bounded number of supporting multi-hop neighbors. This deterministic retrieval stage is important: the semantic truth and truth status of Board entries remain unchanged, while Context Projection decides only which already-recorded evidence is useful for the current local search.

The same principle applies after a Solver has started. For each active Assignment, Context Projection renders clearly relevant Board updates as compact Context Patches, omits clearly irrelevant ones, and refers only ambiguous updates to a transient relevance judge. That judge can decide delivery but cannot invoke tools, create Assignments, or modify the Board. Each Assignment records the Board revision and cursor it has consumed, allowing the Board to grow continuously without forcing a Solver to reread its entire history. Context cost therefore scales with the relevant neighborhood, not the cumulative size of the whole scheme.

5.2.3 Runtime Isolation with Board-Mediated Collaboration

The final obstacle is allowing several local searches to contribute to one scheme without turning their private reasoning into another unbounded shared transcript. SYSSPEAR executes every Assignment in an isolated *Solver*, with an independent model session, working context, and runtime state. Solvers working on the same scheme do share one narrow operational exception, a workspace for reusable files such as scripts or captured requests, but a file there carries no semantic weight until a Solver advertises it through a Board artifact or capability entry. Every other form of collaboration passes through the Strategy Board: a Solver publishes structured updates—a fact, hypothesis, attempt, capability, or artifact, together with its truth status, explicit relations to earlier entries, and any supporting artifact—while other Solvers receive only the related Board evidence selected by Context Projection, never an opaque end-of-session summary.

This Board-mediated collaboration has three properties. First, it preserves attribution and permits disagreement: competing claims coexist as explicit *supports* or *contradicts* relations rather than being overwritten by the most recent report. Second, versioned Board updates ensure that concurrent modifications remain ordered and that stale updates are rejected. Third, isolation makes failure recoverable: a failed executor can be replaced without losing the evidence it produced, while every cross-executor dependency remains inspectable in the single Strategy Board.

5.3 SecFlow: Knowledge-Enhanced Agent Runtime

The runtime isolation mechanism of the previous section presupposes a substrate that can deploy, isolate, and manage many Solvers concurrently, and an expertise base that guides their decisions. Both are difficult to build. Thus, the second challenge is how to make agent execution knowledge-



Figure 8: An attack instance generated by SecFlow using expert knowledge skills. The dependency graph guides the agent through a multi-stage campaign. Each skill’s preconditions are matched against Board evidence, and newly produced postconditions unlock subsequent skills.

able, reliable, and isolated, so that the Board’s guarantees of attribution, versioning, and evidence preservation extend to both the expertise guiding decisions and the physical runtime executing them. On the infrastructure side, a realistic campaign may spawn dozens of Assignments across heterogeneous model backends, and each Solver’s session is long-running, stateful, and prone to interruption by rate limits, timeouts, or crashes. On the knowledge side, an LLM agent operating without domain-specific expertise hallucinates exploit steps, misses business-specific misconfigurations, and cannot chain techniques in the correct order. Naively binding the platform to one agent runtime couples the entire system to that runtime’s failure modes and API surface. A crash mid-Assignment erases not just the model transcript but also the task’s position in the Board. Concurrent Solvers sharing one process or one workspace interfere through resource contention, file races, and cross-session state leakage.

Therefore, we propose *SecFlow*, which is the execution substrate of SYSSPEAR, to address this challenge through three complementary pillars. First, *knowledge enhancement*: over 200 expert skills distilled from more than a decade of security operations across 30,000+ production machines are codified into a structured library with a dependency graph, so the agent receives real-world attack expertise rather than relying on parametric memory. Second, *capability orchestration*: a uniform agent protocol (ACP) mediates all interaction between the platform and heterogeneous runtimes, so Claude Code, Codex, Goose, or a future runtime can be substituted without touching the Board, the Coordinator, or Context Projection. Third, *performance and safety*: the file system serves as the authoritative source of task state, workspace-level isolation prevents concurrent tasks from interfering, and event-driven synchronization propagates state changes without polling—together ensuring that execution is crash-recoverable, concurrently safe, and observably consistent.

5.3.1 Knowledge-Enhanced Attack Expertise

The first obstacle is that an LLM agent operating without domain-specific expertise hallucinates exploit steps, misses business-specific misconfigurations, and cannot chain techniques in the correct order. SecFlow therefore codifies over 200 expert skills, distilled from more than a decade of security operations across 30,000+ production machines, into a structured library that the agent consults at each decision point. Each skill is a self-contained procedure with a name, a precondition over the Board’s evidence types, an ordered list of concrete commands or API calls, and a postcondition describing the evidence a successful execution should produce. For example, a skill named *SSH-key lateral movement* requires a Board entry of type `credential` with `type=ssh-key` and a `reachable-host` entry for the target. It prescribes extracting the key, attempting connection, and publishing either a new `session` entry or a `failed-attempt` entry with the error. A skill named *Log4Shell exploitation* requires a `service` entry with `product=log4j` and a `reachable-port`. It

prescribes a JNDI payload, an out-of-band listener, and publishes a session or vulnerability entry.

These skills are not generic penetration-testing checklists. They encode business-specific attack paths observed across 30,000+ machines. One skill targets exposed Docker API sockets, a misconfiguration found repeatedly in production clusters, that enables container escape and host filesystem access. Another exploits misconfigured Kubernetes RBAC bindings, chaining `kubectl auth can-i` reconnaissance with `kubectl create pod` for privilege escalation to cluster-admin. A third abuses cloud metadata endpoints such as `169.254.169.254` to extract instance credentials, then uses those credentials to enumerate and access cloud storage. These paths are difficult for a model to discover from first principles, because they require knowing not just that a vulnerability exists, but which sequence of actions converts it into a usable capability within a specific infrastructure context.

The remaining challenge is ensuring that the agent attempts skills in an order that respects their real-world prerequisites. To address this, skills are organized through a dependency graph that encodes prerequisite and output relations. A skill's preconditions reference evidence types that other skills produce as postconditions, so the graph makes explicit which skills must succeed before others become applicable. For example, *credential dumping from LSASS memory* requires a session with `privilege=admin`, which in turn requires *privilege escalation via SUID binary* or *kernel exploit*, which in turn requires *initial shell access*. The Coordinator uses this graph to filter candidate skills against the Board's current evidence, presenting only skills whose preconditions are satisfied to the Solver. This prevents the common failure mode of an LLM attempting an exploit before establishing the prerequisite access, or trying lateral movement before obtaining valid credentials. When a Solver publishes new evidence, the dependency graph is re-evaluated to surface newly applicable skills, so the agent's capability set evolves with the Board's state rather than remaining fixed at the start of the campaign.

5.3.2 Capability Orchestration via Protocol-Mediated Lifecycle

The second obstacle is that binding the platform to one agent runtime couples execution reliability to that runtime's failure modes and API surface. SecFlow therefore introduces ACP as a uniform interaction protocol that decouples reasoning execution from task control. The platform never invokes a model API directly. Instead, an `AcpBridge` starts or resumes an Agent Session through ACP, which abstracts session lifecycle, tool invocation, permission requests, and subtask delegation behind a stable, runtime-agnostic interface. A Profile and Mode Router selects the appropriate agent profile and execution mode before the session starts, so the same Assignment can be dispatched to different runtimes without changing the Coordinator's logic or the Board's schema. Specifically, SecFlow uses Claude Code [3] for code-intensive exploitation and Codex [39] for reasoning-heavy reconnaissance.

This protocol-mediated lifecycle preserves both uniform capability extension and declarative session recovery. New tools, MCP servers, or skill definitions are registered through ACP, not through per-runtime adapters, so the platform's capability surface grows without multiplying integration code. Because ACP sessions are resumable, an interrupted Solver can be restored to its last consistent state by replaying the ACP session from the file-system transcript, rather than being restarted from scratch. The platform thus treats heterogeneous runtimes as interchangeable execution backends, while the Strategy Board retains its runtime-agnostic view of evidence and assignments.

5.3.3 Performance and Safety via File-System-Grounded Execution

The final obstacle is ensuring that concurrent execution is both performant and safe. Long-running tasks crash, concurrent Solvers interfere, and state changes must propagate without inconsistency. SecFlow addresses all three through a single design principle, that the file system is the authoritative state, from which crash recovery, concurrency isolation, and real-time consistency follow as consequences rather than separate mechanisms.

A long-running agent task may be interrupted by rate limits, timeouts, or process failures. A crash must not erase the task's position in the Board. To overcome this, SecFlow makes the file system the authoritative source of task state. Every task's description, message queue, event stream, execution record, and analysis report is persisted to disk as it is produced, so the file system *is* the state, not

merely a log of it. A TaskStateManager generates periodic snapshots and JSON Patches, so the full evolution of a task is reconstructable from disk alone. Crucially, the task state machine, which governs the Assignment’s lifecycle, is separated from the session state machine, which governs one ACP session. When a session crashes, the task state is unaffected. The Assignment remains in its last persisted phase, with its Board entries and checkpoint evidence intact. Recovery is local. The AcpBridge resumes or replaces the session, the TaskRuntime replays the pending message queue, and the Solver continues from the last consumed Board revision.

Beyond recovery, concurrency itself poses a hazard. Solvers sharing one process or one workspace interfere through resource contention, file races, and cross-session state leakage. To prevent this, SecFlow adopts a layered process architecture consisting of Gateway, WorkspaceHost, Task, and Agent Runtime, where each layer enforces a distinct isolation boundary. Resource scheduling is workspace-scoped. When concurrent tasks exceed the workspace’s resource budget, new tasks enter a FIFO waiting queue rather than degrading running ones. An idle-session swap mechanism evicts a Solver’s agent process after a threshold idle period, persisting its session state to disk for on-demand restoration. This allows the platform to oversubscribe agent processes relative to physical resources, running more concurrent Assignments than available agent slots without losing any Solver’s progress. Workspace isolation further ensures that a failed or runaway executor cannot corrupt another task’s files, state, or Board evidence.

Even when tasks are isolated and individually recoverable, their state changes must reach the Coordinator, the Board, and human observers without polling and without inconsistency. To achieve this, SecFlow designs a unified Inbox message entry point that routes all messages from the user, from subtasks, from team agents, and from scheduled triggers as file events. When a Solver publishes a Board update, the update is a file-system write. The TaskRuntime observes the write, advances the task state machine, and notifies the AcpBridge if the session needs to act. For external observers, an EventMux multiplexes these file events into Server-Sent Events streams delivered to the Web UI. The synchronization chain, from agent execution to file-system write to JSON Patch to EventMux to SSE to frontend, ensures that what the observer sees is exactly what the agent produced, with no intermediate cache to become stale. This matters for the Strategy Board. When a Solver publishes evidence, the Coordinator can observe the Board update through the same event chain and schedule the next Assignment in response, closing the loop between evidence production and task scheduling without a separate notification mechanism.

5.4 Static-Analysis-Verified Attack Scheme Correctness and Safety

After generating attack scheme, The following challenge is to verify that each scheme is correct in its dependency structure and safe in its software objects before execution. Incorrectness arises when the scheme’s step dependencies violate real-world prerequisites. The agent might schedule an exploit before obtaining prerequisite access, or chain techniques in an impossible order where no valid execution sequence exists. Unsafety arises when the scheme uses software objects that contain vulnerabilities. A vulnerable exploitation tool could compromise the attacker’s own infrastructure, and a targeted service might harbor unknown vulnerabilities that make the attack unpredictable. Detecting these errors by trial execution is costly and potentially destructive.

To address this challenge, SYSSPEAR performs static-analysis-verified checking along these two dimensions. SYSSPEAR uses ORAX [57] to check the dependency satisfiability of each attack scheme using SMT solving with oracle feedback, ensuring that a valid execution order exists for all skill preconditions and postconditions. After that, EUNOMIA [16] is used by SYSSPEAR to provide semantic-enhanced vulnerability analysis through fine-grained symbolic execution with user-specified searching strategies, improving bug detection by up to three orders of magnitude. Moreover, SYSSPEAR leverages SYMGX [48], which provides a novel analysis model for specialized execution environments such as SGX [19] enclaves, to detect cross-boundary pointer vulnerabilities that conventional approaches miss. Together, these three mechanisms ensure that no attack scheme reaches execution unless its dependencies are formally satisfiable and its software objects are verified safe.

5.4.1 Dependency Satisfiability via SMT Solving

The first challenge is the correctness of the attack schemes generated by the SYSSPEAR. An LLM-generated attack scheme encodes dependencies among skills, where each skill’s preconditions must

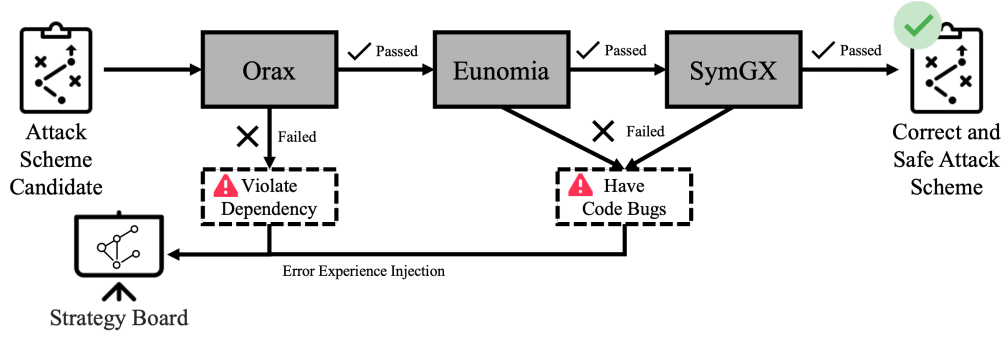


Figure 9: Static-analysis-verified attack scheme checking in SYSSPEAR. ORAX checks dependency satisfiability via SMT solving. EUNOMIA and SYMGX check software safety via fine-grained symbolic execution and SGX-specific analysis.

be satisfied by evidence produced by earlier skills. The agent may produce a scheme where these dependencies are circular, contradictory, or simply unsatisfiable, meaning no valid execution order exists. To overcome this, our insight is to model the attack scheme’s dependency constraints as a Satisfiability Modulo Theories problem. If the constraints are satisfiable, a valid execution order provably exists. If not, the scheme is provably incorrect and must be revised before execution.

A key difficulty in SMT-based verification of attack schemes is that the scheme’s dependencies involve not just logical constraints but also oracle components. External tools, network services, and library functions have observable behavior but unknown implementation, which constitutes an SMTO problem where the SMT solver must reason about constraints involving oracles it cannot fully inspect. ORAX establishes a feedback loop between the SMT solver and the oracle handler. The solver analyzes the deficiency in oracle mapping information it currently has and feeds this back to the handler. The handler then provides the most suitable additional oracle mapping information to the solver. A dual-clustering strategy selects the initial oracle mapping information, and a relaxation-based method analyzes the information deficiency.

In practice, each skill’s preconditions and postconditions are encoded as SMT constraints over the Board’s evidence types. ORAX checks whether there exists a valid execution order that satisfies all constraints simultaneously. If the scheme is unsatisfiable, it is rejected and returned to the Coordinator for revision, preventing the agent from executing a scheme that is doomed to fail.

5.4.2 Semantic-Enhanced Vulnerability Analysis

The second challenge is of the attack schemes from the software vulnerability perspective. An attack scheme may use software objects such as exploitation tools, scripts, and libraries that contain vulnerabilities. Using a vulnerable tool could compromise the attacker’s own infrastructure or leak attack artifacts to the defender. Existing symbolic execution approaches use coarse-grained global searching strategies that cannot efficiently traverse complex code structures, leading to path explosion and missed vulnerabilities. To overcome this, our insight is that different code regions have different structural characteristics, so searching strategies should be localized and user-specified rather than globally uniform, allowing the analysis to prioritize paths relevant to the attack context while avoiding wasted exploration.

Realizing this insight requires a way for users to express local searching strategies concisely and a mechanism to prevent different strategies from interfering through shared variable contexts. EUNOMIA introduces AES, a domain-specific language that lets users specify local searching strategies for different parts of the program. Rather than applying one global strategy, EUNOMIA allows different code regions to use different search priorities, targeting specific paths relevant to the attack context. It isolates the context of variables for different local searching strategies, avoiding conflicts between regions. EUNOMIA is the first symbolic execution engine supporting the full feature set of WebAssembly, enabling analysis of applications written in various languages that compile to WebAssembly. This broad coverage is important because attack tools are often written in different languages and distributed as binaries. In SYSSPEAR, EUNOMIA analyzes the software objects ref-

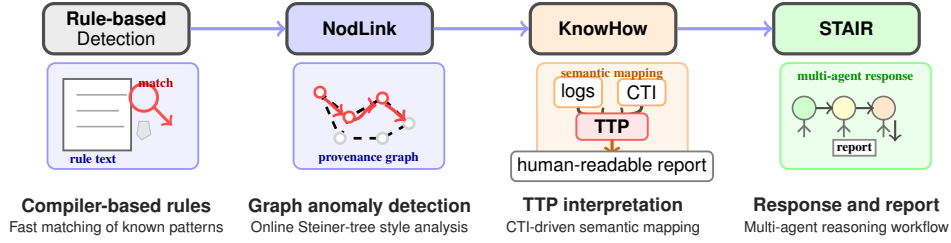


Figure 10: Overview of SYSARMOR.

erenced in an attack scheme before execution, checking for vulnerabilities that could compromise the attack or the attacker.

5.4.3 Model-Driven Vulnerability Analysis for Specialized Environments

The third challenge is the attack scheme’s safety in specialized execution environments. Some attack targets use hardware-assisted trusted execution environments such as Intel SGX enclaves, which have unique vulnerability types not covered by conventional analysis. SGX applications heavily use cross-boundary pointers to exchange data between secure enclaves and untrusted environments. These pointers introduce a vulnerability class that existing symbolic execution approaches cannot detect, due to three unique features of SGX applications. Multi-entry arbitrary-order execution allows ECalls in any order. Stateful execution maintains enclave state across calls. Context-aware pointers change their semantics depending on whether they originate from inside or outside the enclave. To overcome this, our insight is that SGX applications require a dedicated analysis model that natively captures these three execution features, rather than retrofitting conventional models that were not designed for enclave semantics.

Realizing this insight requires modeling all three features simultaneously in a single analysis framework, which conventional symbolic execution engines were not designed to support. SYMGX introduces a Global State Transition Graph with context-aware analysis that properly handles the three unique features. The graph models multi-entry arbitrary-order execution by tracking all possible ECall sequences. It captures stateful execution by maintaining enclave state across transitions. It handles context-aware pointers by distinguishing pointer origins and applying different validation rules. This model enables SYMGX to detect cross-boundary pointer vulnerabilities that conventional approaches miss entirely.

In SYSSPEAR, SYMGX analyzes SGX-protected targets in the attack scheme, identifying cross-boundary pointer vulnerabilities that the attacker could exploit. This ensures that the scheme’s approach to SGX targets is based on verified vulnerability information rather than assumptions, and that the attacker does not inadvertently trigger unintended enclave behavior that could compromise the attack.

6 SYSARMOR

SYSARMOR is a provenance-driven endpoint detection and response system that integrates rule-based detection, graph-based anomaly detection, and TTP-level reasoning into a unified architecture. As shown in Figure 10, the system is organized into multiple components that collectively support scalable monitoring, real-time alert generation, and post-detection investigation over large-scale host telemetry.

At the core of the design is a distributed processing engine built on Apache Flink, which provides the runtime substrate for continuous analysis over high-volume provenance streams. On top of this pipeline, SYSARMOR combines four complementary detection modules: a Falco-compatible rule engine for fast matching of known attack patterns, NODLINK [29] for identifying suspicious structural deviations in provenance graphs, KNOWHOW [32] for mapping suspicious behavior to high-level Tactics, Techniques, and Procedures (TTPs) techniques, and STAIR [20] for higher-level reasoning and investigation support. By coupling signature-based precision with graph-based anomaly detection and semantic interpretation, SYSARMOR is able to detect both known and emerg-

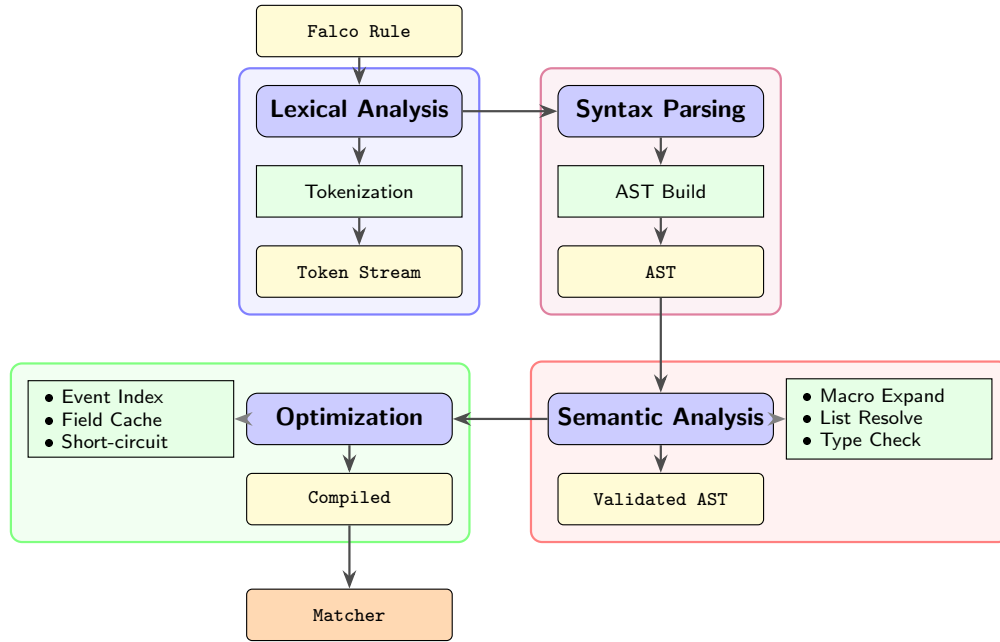


Figure 11: Rule compilation pipeline with four phases

```

rule suspicious_tmp_execution {
  condition:
    (evt.type in (execve, execveat) and
     (proc.exe startswith "/tmp/" or
      proc.exe startswith "/dev/shm/"))
}

```

Figure 12: Example Falco rule `suspicious_tmp_execution` for detecting process executions from temporary directories.

ing Advanced Persistent Threats (APT) behaviors while preserving interpretability for security analysts.

6.1 Rule-based Detection

SYSARMOR incorporates a compiler-based rule engine that preserves the familiar Falco rule syntax while introducing compiler-style optimization for high-throughput detection. The system first compiles rule specifications into efficient internal representations, and then evaluates the compiled logic against normalized streaming events with low-overhead matching procedures. This design allows the platform to retain rule expressiveness for security analysts while substantially reducing runtime cost under large-scale telemetry workloads.

Compilation Phase. Figure 11 summarizes the four-stage compilation pipeline used by the rule engine. During this stage, Falco rules are parsed into a structured Abstract Syntax Tree (AST) that preserves logical semantics while enabling efficient execution. The process consists of lexical analysis and AST construction, which translate rule conditions into a machine-readable form; semantic analysis, which expands macros and resolves symbolic references; and a final optimization pass that improves runtime efficiency by indexing event types, caching frequently accessed fields, and exploiting short-circuit evaluation.

Lexical analysis converts the rule into a sequence of atomic tokens. *BinaryOpNode*("and", *In*(*FieldNode*("evt.type"), *SetNode*({"execve", "execveat"})), *BinaryOpNode*("or",

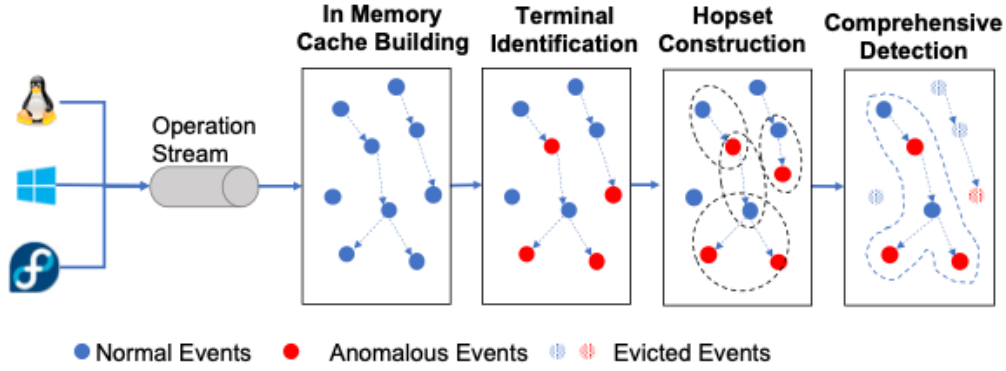


Figure 13: Overview of the NODLINK pipeline. Normal provenance events are ingested from the operation stream, cached in memory, and progressively expanded from suspicious terminal nodes into a hopset for comprehensive attack detection.

StartsWith(FieldNode("proc.exe"), "/tmp/"), StartsWith(FieldNode("proc.exe"), "/dev/shm/"))
 The semantic analysis phase then validates the AST and resolves references to schema fields, macros, and list definitions. It recursively expands macro expressions, materializes list values, and checks type compatibility and field existence against the event schema. Only well-formed rules proceed to optimization, preventing undefined references or malformed predicates from reaching the runtime matcher.

The optimization stage further improves execution efficiency through several mechanisms. Event-type indexing builds a direct mapping from event categories to relevant rules, allowing the system to filter candidate matches before evaluating predicates. In a naive rule engine, each event may need to be checked against all N rules, yielding $O(N)$ matching cost. In contrast, the compiler extracts event constraints from each rule, such as 'evt.type = execve', and uses a hash lookup at runtime to retrieve only the relevant subset $R_t \subset R$ for an incoming event type t , where $|R_t| \ll |R|$. In addition, cached field access avoids repeated parsing of the same event attributes, while short-circuit evaluation terminates logical expressions as soon as the outcome is determined.

Rule Matching Phase. The matching phase executes within the streaming detection pipeline and evaluates compiled rules against normalized events in real time. At runtime, an incoming event first consults the event-type index to obtain a candidate rule set, then extracts and caches any required fields in the event context. Predicates are evaluated in a cost-aware order and benefit from short-circuit semantics to avoid unnecessary computation. Once a rule is satisfied, the engine applies suppression and rate-control policies, assembles a structured alert containing the rule identifier, matched fields, and provenance context, and forwards the result to the Alerts Kafka topic and OpenSearch [13] for storage and subsequent investigation.

6.2 NodLink

NODLINK is the graph-based anomaly detection component of SYSARMOR. It formulates online APT detection as an Online Steiner Tree Problem (OSTP) over provenance graphs. The detector maintains a compact evolving graph, identifies suspicious terminal nodes, and incrementally expands them into a hopset that preserves the causal structure of attack-related activities. By reasoning over the relationships among subjects, objects, and execution paths rather than over isolated events, NODLINK is able to expose stealthy attacks that unfold through a sequence of seemingly benign actions.

Figure 13 summarizes the main stages of the pipeline. The operation stream denotes the normalized provenance input produced by upstream Flink jobs. During in-memory cache building, recently observed events and graph fragments are retained so the detector can update its state incrementally without rescanning the full history. This cache also enforces bounded-memory execution by evicting stale events that fall outside the active detection window.

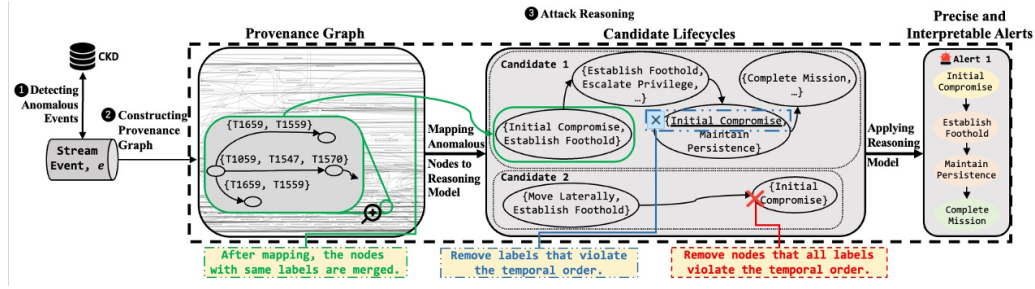


Figure 14: Overview of the KNOWHow pipeline. Suspicious provenance subgraphs produced by NODLINK are matched against CTI-derived knowledge, organized into candidate attack lifecycles, and refined into precise and interpretable alerts.

The terminal identification stage assigns anomaly scores to nodes that are likely to serve as attack endpoints. In the figure, the blue backbone represents normal provenance structure, whereas the red nodes indicate suspicious terminals. These terminals are not treated as the final detection result; instead, they act as anchors for the subsequent graph expansion. The hopset construction stage then traverses the causal neighborhood around each suspicious terminal and forms a compact subgraph that connects the relevant attack entities while preserving provenance semantics. The dashed outlines in the figure indicate the progressively expanded suspicious region, and the evicted events in the legend correspond to cached events that are removed once they are no longer useful for the current window.

In the final comprehensive detection stage, NODLINK evaluates the expanded hopset to determine whether it contains a coherent attack pattern. This stage aggregates evidence from the suspicious terminals, their surrounding causal paths, and the retained cache state to produce a single detection outcome. The resulting provenance subgraph can then be forwarded to the downstream alerting pipeline and combined with rule-based detections. This incremental design avoids heavy retraining, supports real-time inference, and provides stronger contextual evidence than event-level matching alone.

6.3 KnowHow

KNOWHow complements NODLINK by providing semantic reasoning over the suspicious provenance subgraphs identified by the anomaly detector. It converts a structurally suspicious graph into a sequence of candidate attack lifecycles and then reasons over those candidates using cyber threat intelligence. Rather than returning only a binary anomaly label, KNOWHow explains which adversarial behaviors the observed provenance is most consistent with and produces TTP-level outputs that are directly useful for investigation.

Figure 14 shows the main stages of the pipeline. The provenance graph on the left is the graph-level input passed from NODLINK. At this stage, suspicious nodes and edges already encode the contextual evidence produced by the graph anomaly detector. The candidate-lifecycle stage then constructs one or more attack hypotheses, each corresponding to a plausible progression of attack stages such as initial compromise, establishment of foothold, lateral movement, privilege escalation, and mission completion. During this stage, KNOWHow also applies temporal-order constraints to prune inconsistent hypotheses: labels that violate the expected progression are removed, and nodes that cannot be placed into a valid lifecycle are discarded. This filtering step is important because it prevents the reasoning model from overfitting to isolated graph fragments that are structurally suspicious but semantically implausible.

In the attack-reasoning stage, KNOWHow performs semantic alignment between the candidate lifecycles and structured knowledge extracted from Cyber Threat Intelligence (CTI) reports. The module relies on General Indicator of Compromise (gIoC) representations, which encode adversary knowledge as subject-verb-object triples. These triples provide a compact bridge between natural-language threat intelligence and provenance observations, allowing the system to match CTI-derived behaviors against the suspicious graph. The reasoning model then selects the candidate lifecycle that best satisfies both the graph evidence and the semantic constraints encoded in the knowledge base.

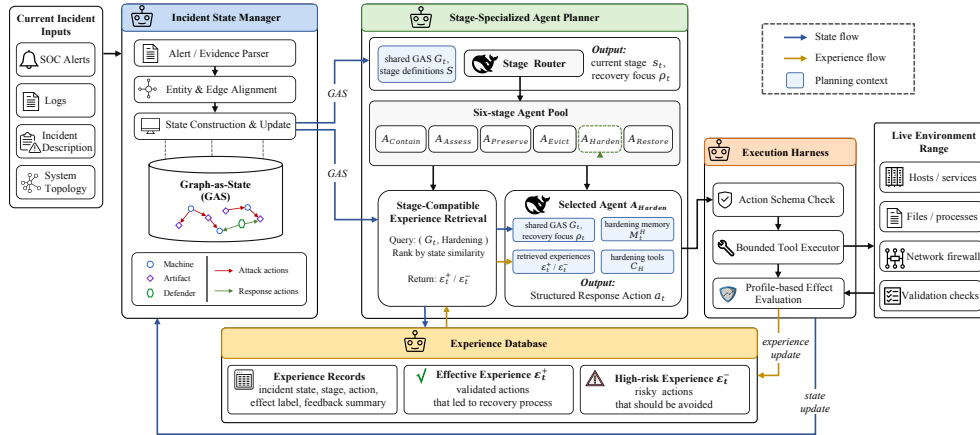


Figure 15: Overview of STAIR. STAIR maintains an evolving incident state with Graph-as-State (GAS), performs stage-aware response planning with historical experience, and incorporates execution feedback to form a closed-loop automated response process.

The final output is a precise and interpretable alert. As illustrated in the rightmost panel, the selected lifecycle is converted into an attack narrative that summarizes the most likely stage sequence and the associated TTPs labels. This representation gives analysts both the structural evidence from the provenance graph and the semantic explanation from CTI, making it easier to understand the attack progression and to prioritize response actions. In this way, KNOWHOW serves as a reasoning layer that turns graph anomalies into actionable, human-readable detection results.

6.4 STAIR

STAIR is the automated incident-response component of SYSARMOR. As shown in Fig. 15, it organizes incident response as a closed-loop process that continuously tracks incident state, determines the current recovery stage, generates response actions, and incorporates execution feedback. STAIR targets two major challenges in long-horizon response. First, incident states evolve continuously as attacks and defensive actions change the environment, making previously handled entities, unresolved attack paths, and recovery progress easy to lose over long interaction histories. Second, different response stages pursue different operational objectives, and an action that is reasonable in one stage may be inappropriate in another. These two problems lead to *state loss* and *stage mismatch*, respectively.

To preserve response continuity, the Incident State Manager represents the evolving incident as a *Graph-as-State* (GAS). GAS jointly captures affected machines and artifacts, attack behaviors, response actions, and their execution results. It is initialized from SOC alerts, logs, and system information, and is incrementally updated after every response action using execution feedback. This persistent representation allows subsequent planning to operate on the current incident state rather than repeatedly reconstructing progress from textual interaction history.

Based on the maintained GAS, the Stage Router identifies the current recovery stage and its concrete recovery focus. STAIR organizes response into six stages—CONTAINMENT, ASSESSMENT, PRESERVATION, EVICTION, HARDENING, and RESTORATION—and dispatches planning to the corresponding stage-specialized agent. The selected agent reasons with the shared GAS, stage-compatible historical experience, and stage-scoped response tools, thereby constraining action generation to the current recovery objective.

The resulting structured action is validated and executed by the Execution Harness. Execution feedback is written back to GAS and used to update reusable response experience, forming a continu-

ous state-planning-execution loop. By combining persistent state tracking with stage-aware planning, STAIR reduces state loss and stage mismatch in long-horizon incident response and extends SYSARMOR from attack detection and interpretation to automated response.

7 Evaluation

7.1 Evaluation of SYSFIELD

We evaluate SYSFIELD through four questions that follow the measurement pipeline from the quality of the collected evidence to the difficulty of the generated ranges and, finally, to the interpretation of agent behavior:

- **RQ1: Measurement Fidelity and Cost.** Can SYSFIELD collect complete runtime evidence without imposing prohibitive system overhead?
- **RQ2: Range Difficulty and Scale.** Do multi-step composition and larger topologies expose capability gaps that are hidden by single-step or small-range evaluations?
- **RQ3: Multi-stage Progression under Controlled Conditions.** Where do current cyber agents lose progress, and how does the current high-interference configuration affect downstream completion and execution cost?
- **RQ4: Offensive Progression and Defensive Evidence.** How does independently verified attack completion relate to runtime observability, and which expected behaviors are least consistently represented in the evidence?

Together, these questions test whether the evaluation substrate is trustworthy, whether the constructed ranges provide meaningful difficulty, and whether the resulting trajectories support fine-grained offensive and defensive analysis.

7.1.1 Evaluation Setup

Dataset. The primary trajectory dataset is Stratified-50: 50 three-tier enterprise cases constructed from 24 unique CVEs and stratified by historical entry-stage and intermediate-stage difficulty across six cells (easy/easy, easy/hard, mid/easy, mid/hard, hard/easy, and hard/hard). Each case contains three ordered CVE slots and three independently verified flags: a perimeter entry service (target 1), an intermediate application service (target 2), and a data service (target 3). Historical results determine sampling strata only and are not reused as model outcomes. CVEs recur across cases, and the current data layer contains only three CVE variants; the 50 cases are therefore not statistically independent vulnerability samples or a complete representation of enterprise attack paths [6].

We additionally report substrate-level measurements that are not part of Stratified-50: collector stress tests, presentation-level aggregate comparisons between single-step CyberGym tasks and SYSFIELD multi-step ranges, a 7-node versus 50-node scale comparison, and construction-capacity measurements. These sources have different units and controls and are analyzed separately from the case-level trajectory data.

Metrics. Measurement fidelity is quantified by event-loss rate and additional system overhead. Range difficulty is reported as verified attack success and the percentage-point change under multi-step composition or larger topology scale. Trajectory capability is measured by target-1, target-2, and target-3 completion, all-three completion, business-objective completion, deepest stage, termination status, timeout, and runtime. Defensive observability is measured independently by the presence of any new attack-window signal, strict coverage of every expected rule, per-rule missing-signal counts, and the four attack-success/signal-hit quadrants. Signal-frame volume is retained as observation volume, not interpreted as a count of independent behaviors or as a severity score.

Baselines. The collector-completeness baselines are Sysdig [46] and LTTng [31]; the overhead baselines are HARDLOG [2] and OMNILOG [14]. They belong to different experiments and are not combined into one collector ranking. The range-difficulty baselines are single-step CyberGym [50] and the reported 7-node SYSFIELD configuration, each compared with its corresponding multi-step or 50-node condition for Kimi K3 [26], GLM-5.2 [56], DeepSeek-V4-Pro [8], and GPT-5.6 Luna [40]. The trajectory baselines are the single-agent Kimi K3 and DeepSeek-V4-Pro L2 runs.

The same-backbone SYSSPEAR analysis uses the DeepSeek-V4-Pro single-agent L2 arm as its reference, and the interference analysis compares the recorded DeepSeek-V4-Pro L1 none and high configurations.

Setup for Baselines. Table 5 states what is held fixed and what changes in each comparison. This prevents a presentation-level baseline, a same-protocol model comparison, and a configuration-level system comparison from being interpreted as if they were one randomized experiment.

Table 5: Baseline setup and comparison boundaries for the SYSFIELD evaluation.

Comparison	Shared setup	Changed factor and interpretation
L2 model comparison	Same 50 cases, OpenAI-compatible SDK and runner, L2 context, tools, 300-turn budget, 3,600-s agent timeout, serial execution, verifier, and SYSARMOR configuration	Kimi K3 versus DeepSeek-V4-Pro; same-protocol descriptive model comparison, with one trajectory per model-case pair
Same-backbone attack system	Same 50 cases, DeepSeek-V4-Pro, public L2 scenario, 300-turn budget, 3,600-s timeout, serial execution, private verifier, and observe-only SYSARMOR	Single-agent runner versus SYSSPEAR assessment/session adapter; configuration-level comparison because the runner and execution policy change together
L1 interference	Same manifest, DeepSeek-V4-Pro, L1 context, 300 turns, 3,600-s timeout, seed 1, and temperature 0	none: parallelism 8; high: parallelism 4, 43 decoys, and one omitted topology hint, in fixed order; comparison of recorded configurations, not a pure decoy effect
Collector completeness	Same stress-oriented event-generation and accounting protocol on the reported hardware/virtualization settings	SYSFIELD versus Sysdig and LTTng; event-loss comparison only
Collector overhead	Same reported workload family within the overhead experiment	SYSFIELD versus HARDLOG and OMNILOG; overhead comparison only
Composition and scale	Same model within each reported pair	CyberGym single-step versus SYSFIELD multi-step, and 7-node versus 50-node; source reports omit trial-level uncertainty and some controls, so deltas remain protocol-specific

Across the Stratified-50 arms, environment validation, attack-graph validation, path reachability, and cleanup succeed for all formal cases, so the differences analyzed below occur after range construction and qualification. The direct L2 model arms each contain one recorded trajectory per case. The L1 comparison changes parallelism and one topology hint together with the decoy condition, and the SYSSPEAR comparison changes the attack-system runner and execution policy. We therefore use these results as descriptive evidence under their recorded protocols rather than as universal rankings or component-level causal effects.

Implementation and Experiment Environment. SYSFIELD is implemented on the CVELab range-generation and evaluation stack. The stack uses Python 3.12, uv, Docker, ContainerLab 0.74 or later, and Ansible. Initialization consists of resolving the Python environment with uv, building the attacker and per-CVE runtime images, generating an `enterprise_3tier` scenario from a case manifest, deploying its container network, applying the Ansible setup, and waiting for service readiness. Each formal run then launches the attack system, verifies stage flags and the business objective through a model-independent verifier, preserves the structured run artifacts, and destroys the range. Workload nodes communicate through the emulated data plane rather than the Docker management network; only the attacker-side route retains the egress required to call the model endpoint. SYSARMOR is enabled as an observe-only monitor, so it records signals without blocking the attack [6].

All case-level Stratified-50 model arms use pretrained inference endpoints through an OpenAI-compatible interface; SYSFIELD performs no fine-tuning or task-specific training. Consequently, there is no training split, optimizer, epoch count, or learning-rate setting to report. Inference controls are stated per arm below rather than treated as global hyperparameters. The range experiments use isolated Docker/ContainerLab topologies; the separate collector stress tests cover virtual and physical machines under both 1-CPU/2-GB and 32-CPU/64-GB configurations. The available artifacts do not disclose a single host-CPU model or OS build shared by every range run, so we do not introduce one as an experimental control.

7.1.2 RQ1: Measurement Fidelity and Cost

Table 6 summarizes the reported results for the DPUAudit-backed SYSFIELD collector [23]. This configuration records all generated events in every reported hardware and virtualization setting, yielding a 0% event-loss rate. Under the same stress-oriented completeness comparison, Sysdig loses 98.5% of events and LTTng loses 60.0%. The result matters for evaluation validity: when most events are dropped, an apparent absence of attack behavior may be a collector artifact rather than a property of the agent trajectory.

Table 6: Reported completeness and overhead of the DPUAudit-backed SYSFIELD collector. Completeness and overhead use different baseline sets.

Metric	DPUAudit-backed SYSFIELD	Reported baselines
Event-loss rate	0.0%	Sysdig: 98.5%; LTTng: 60.0%
Additional system overhead	2.1%	HARDLOG: 23.2%; OMNILOG: 33.0%

The completeness gain does not come from accepting a large execution penalty. The DPUAudit-backed SYSFIELD configuration adds 2.1% overhead, compared with 23.2% for HARDLOG and 33.0% for OMNILOG. Relative to these two reported baselines, the overhead is lower by 90.9% and 93.6%, respectively, which explains the 90–94% reduction summarized in the presentation. The event-loss and overhead experiments use different baselines and should not be collapsed into a single multi-metric ranking; jointly, however, they show that the framework can preserve the evidence needed for later trajectory analysis while remaining close to the reported 3% deployment budget.

7.1.3 RQ2: Range Difficulty and Scale

The first difficulty comparison changes the evaluation from single-step vulnerability reproduction in CyberGym to multi-step attack execution in SYSFIELD. Table 7 shows a consistent reduction for all four evaluated models. Success falls by 18.1 percentage points for Kimi K3, 21.6 points for GLM-5.2, 21.6 points for DeepSeek-V4-Pro, and 20.8 points for GPT-5.6 Luna. Thus, the reported 18–22 point gap is not caused by one outlying model. Requiring the agent to preserve state and satisfy dependencies across several attack steps exposes failure modes that a single-step endpoint does not measure.

Table 7: Reported attack success on single-step CyberGym tasks and SYSFIELD multi-step ranges.

Model	CyberGym single-step	SYSFIELD multi-step	Change
Kimi K3	73.8%	55.7%	−18.1 pp
GLM-5.2	62.8%	41.2%	−21.6 pp
DeepSeek-V4-Pro	57.8%	36.2%	−21.6 pp
GPT-5.6 Luna	44.3%	23.5%	−20.8 pp

Topology scale produces a second, separate reduction. As shown in Table 8, increasing the range from 7 to 50 nodes lowers attack success by 9.8–10.6 percentage points across the same four models. The model ordering is unchanged, but every model loses roughly ten points. This consistent shift suggests that the larger search space and longer dependency structure add difficulty even when the evaluated model family changes.

Table 8: Reported attack success as SYSFIELD range size increases from 7 to 50 nodes.

Model	7-node range	50-node range	Change
Kimi K3	60.8%	50.2%	−10.6 pp
GLM-5.2	46.1%	36.0%	−10.1 pp
DeepSeek-V4-Pro	40.9%	30.6%	−10.3 pp
GPT-5.6 Luna	29.0%	19.2%	−9.8 pp

These difficulty results are supported by the construction-scale measurements. SYSFIELD reports orchestration support for more than 257 CVEs and 1,148 multi-vulnerability ranges. Under a 256-GB memory limit, the platform supports ranges with more than 300 system nodes and campaigns with more than 150 attack behaviors. In the reported commercial-LLM trials, no model escaped the isolated range or bypassed the external verification mechanism. This is evidence about the tested configurations, not a general proof of containment against arbitrary adversaries. The presentation additionally reports that combining SYSFIELD ranges with active SYSARMOR defenses reduces SOTA-agent success by more than 60%; because that experiment enables blocking, we keep it separate from the observe-only defensive analysis in RQ4.

The two tables establish complementary forms of difficulty: multi-step composition expands the required attack process, whereas topology growth expands the environment through which that process must be executed. The presentation does not provide the trial counts, uncertainty estimates, or all controls behind these aggregate comparisons, so the reported deltas should be interpreted within their respective protocols rather than as causal estimates attributable only to step count or node count.

7.1.4 RQ3: Multi-stage Progression under Controlled Conditions

Stage progression. Figure 16 reports verified stage completion on the same 50 L2 cases. Initial access is relatively close: Kimi K3 reaches target 1 in 22 cases, while DeepSeek-V4-Pro does so in 19. The difference becomes larger after the initial foothold. Kimi K3 reaches target 2 in 18 cases and target 3 in 16, whereas DeepSeek-V4-Pro reaches them in 10 and 6 cases, respectively. Thus, the observed model gap is modest at entry but expands as the task requires continued use of an existing foothold and successful transition to later stages.

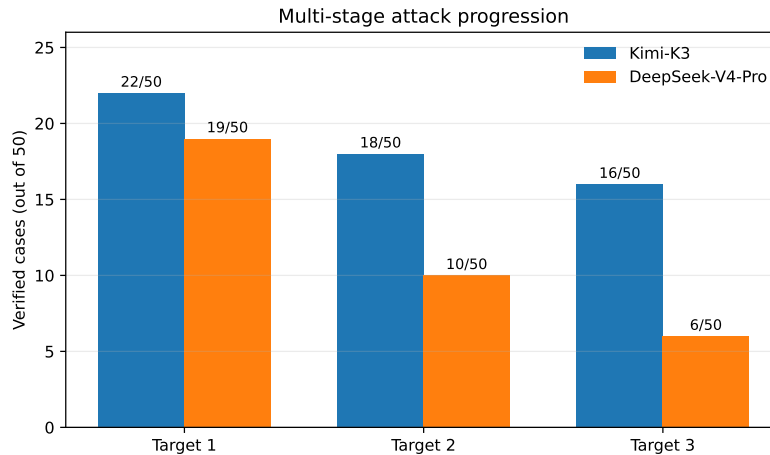


Figure 16: Stage-level completion on the same 50 Stratified-50 L2 cases. Bars report independently verified target flags. The model gap widens after initial access.

The transition rates make this pattern more explicit. After reaching target 1, Kimi K3 continues to target 2 in 18/22 cases (81.8%) and to target 3 in 16/22 (72.7%). DeepSeek-V4-Pro continues to target 2 in 10/19 cases (52.6%) and to target 3 in 6/19 (31.6%). Equivalently, Kimi K3 loses six cases between the first and final stages, whereas DeepSeek-V4-Pro loses thirteen. The primary difference under the current protocol therefore lies in sustained multi-stage progression rather than initial entry alone.

This result is important because later-stage progress requires more than triggering another isolated vulnerability. The agent must retain useful information from earlier steps, operate from the current foothold, identify a reachable continuation target, and recover from failed actions without losing the attack state. Stage-level verification shows where this longer-horizon process breaks down instead of compressing all failures into a single final score.

Same-backbone SYSSPEAR rerun. We additionally evaluate SYSSPEAR on the same 50 Stratified-50 L2 cases using DeepSeek-V4-Pro as the solver backbone. The rerun preserves the CVELab scenarios, public L2 task description, private flag verification, and observe-only SYSARMOR instrumentation, while replacing the attack-side execution path with the SYSSPEAR assessment/session adapter and using `--max-turns 300`. All 50 environment-only validations succeed. Because the attack-system runner and recorded execution policy differ from the single-agent arm, we treat this experiment as a same-backbone, configuration-level comparison rather than a component-level causal ablation.

Table 9: Stage progression under the recorded L2 configurations. SYSSPEAR uses DeepSeek-V4-Pro as its solver backbone; the comparison holds the backbone fixed but changes the attack-system architecture and recorded execution policy.

Attack configuration	Backbone	Target 1	Target 2	Target 3
Single-agent	Kimi K3	22/50 (44%)	18/50 (36%)	16/50 (32%)
Single-agent	DeepSeek-V4-Pro	19/50 (38%)	10/50 (20%)	6/50 (12%)
SYSSPEAR	DeepSeek-V4-Pro	34/50 (68%)	15/50 (30%)	7/50 (14%)

SYSSPEAR reaches target 1 in 34/50 cases (68%), target 2 in 15/50 (30%), and target 3 in 7/50 (14%). Relative to the recorded DeepSeek-V4-Pro single-agent arm, which reaches the three targets in 19, 10, and 6 cases, respectively, the absolute improvement decreases from 30 percentage points at target 1 to 10 points at target 2 and only 2 points at target 3. The improvement is therefore strongly front-loaded: SYSSPEAR substantially expands initial-foothold coverage, but the additional entry successes do not propagate proportionally through the later stages of the attack chain.

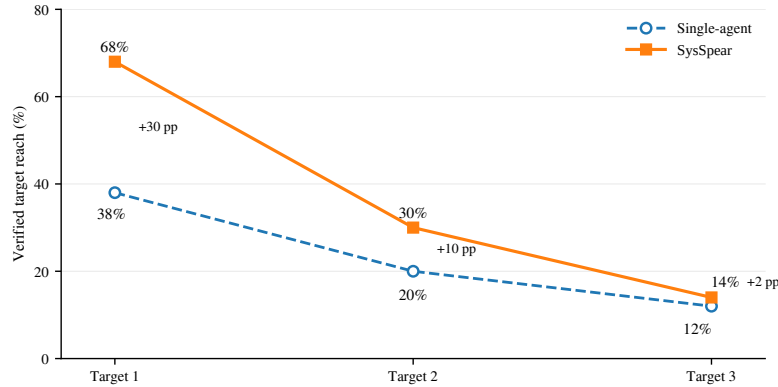


Figure 17: Same-backbone progression profile on the 50 Stratified-50 L2 cases. The gap between the DeepSeek-V4-Pro single-agent arm and SYSSPEAR narrows from 30 percentage points at Target 1 to 10 at Target 2 and 2 at Target 3. Because the comparison changes the attack-system runner and execution policy, it should be interpreted as a configuration-level comparison rather than a component-level causal ablation.

Among the 34 SYSSPEAR executions that reach target 1, 15 (44.1%) continue to target 2, and 7 of those 15 (46.7%) then reach target 3; end-to-end, 7/34 (20.6%) of the initial footholds progress to the final target. Thus, 19 cases lose progress immediately after initial access and another eight stop before the final target. Together with the single-agent results, this shifts the diagnostic focus from vulnerability triggering alone toward post-compromise state utilization, internal target selection, and cross-stage continuation. These conditional rates should not, however, be interpreted as evidence that SYSSPEAR reduces post-foothold capability: SYSSPEAR reaches target 1 on many cases that the single-agent baseline never enters, so a matched comparison over commonly reached footholds is required to isolate the downstream effect.

Termination behavior provides a complementary view of the original matched model comparison. Kimi K3 records 22/50 agent timeouts; none are recorded for DeepSeek-V4-Pro. Deeper progression therefore does not coincide with uniformly lower execution cost or simpler termination behavior. Under the current protocol, Kimi K3 reaches later stages more often but also more frequently

consumes the full time budget. Because there is only one recorded trajectory per model and case, this difference should be treated as a descriptive operating pattern rather than a general reliability ranking.

Environmental interference. The SYSSPEAR rerun above does not complete the predefined matched harness-uplift ablation because the attack-system runner, recorded execution policy, and turn budget change together. The available evidence for environmental dependence comes from the DeepSeek-V4-Pro L1 comparison between the current none and high configurations.

Table 10 summarizes progression, timeouts, and execution cost. Both configurations reach target 1 in 2 of 50 cases. Under none, both cases continue through targets 2 and 3; under high, neither progresses beyond the first stage. The unchanged entry count and the disappearance of downstream completion indicate that the largest observed difference occurs after entry, when the agent must continue exploration and select the correct path through the internal environment.

Table 10: DeepSeek-V4-Pro L1 results under the current none and high configurations.

Metric	none	high
Target 1 / Target 2 / Target 3	2 / 2 / 2	2 / 0 / 0
All-three completion	2/50	0/50
Objective achieved	1/50	0/50
Timeout	6/50 (12%)	19/50 (38%)
Mean agent time	1,417.6 s (23.6 min)	2,428.5 s (40.5 min)
Median agent time	1,091.7 s (18.2 min)	2,536.5 s (42.3 min)
Recorded decoy interaction	–	50/50
Direct decoy contact	–	38/50

The execution-cost measurements support the same interpretation. Timeouts increase from 6/50 to 19/50, a rise of 26 percentage points. Mean agent time increases from 23.6 to 40.5 minutes (71.3%), while median time increases from 18.2 to 42.3 minutes (132.3%). The larger change in the median suggests that the additional cost is not explained only by a small number of unusually long runs; the typical high-configuration run also consumes substantially more of the available execution budget.

The interaction records are consistent with this increase in exploration cost. Every high-configuration run contains a recorded decoy interaction, and 38/50 directly contact a decoy endpoint. This indicates that the agent spends part of its execution interacting with the enlarged target surface. However, the experiment does not isolate decoys as the sole cause: the two arms also differ in worker parallelism, run in a fixed none-then-high order, and contain a topology-hint difference, while decoy interaction is derived from transcript diagnostics rather than packet-level provenance. The supported conclusion is therefore narrower: the *current high configuration* is associated with greater execution cost, more timeouts, and weaker progression beyond the first stage. A causal estimate of deception alone would require matched parallelism and topology hints, randomized arm order, and repeated trials.

7.1.5 RQ4: Offensive Progression and Defensive Evidence

Case-level observability. Figure 18 compares complete attack success with two levels of defensive evidence across the two matched direct-run L2 arms. Kimi K3 completes all three targets in 16/50 cases, produces at least one new attack-window signal in 42/50, and covers all expected rules in 28/50. DeepSeek-V4-Pro completes all three targets in 6/50 cases, while producing a new signal in 30/50 and covering all expected rules in 14/50. For both models, observable runtime behavior is therefore substantially more common than complete compromise.

The difference between the two defensive metrics is itself informative. For Kimi K3, any new signal is observed in 84% of cases, whereas strict expected coverage is 56%; for DeepSeek-V4-Pro the corresponding values are 60% and 28%. The gaps of 28 and 32 percentage points show that “the defense observed some attack-associated behavior” and “the defense observed the complete expected behavior set” are distinct levels of observability. A final success label loses even more process information: any-signal coverage exceeds all-three completion by 52 percentage points for Kimi K3 and 48 points for DeepSeek-V4-Pro.

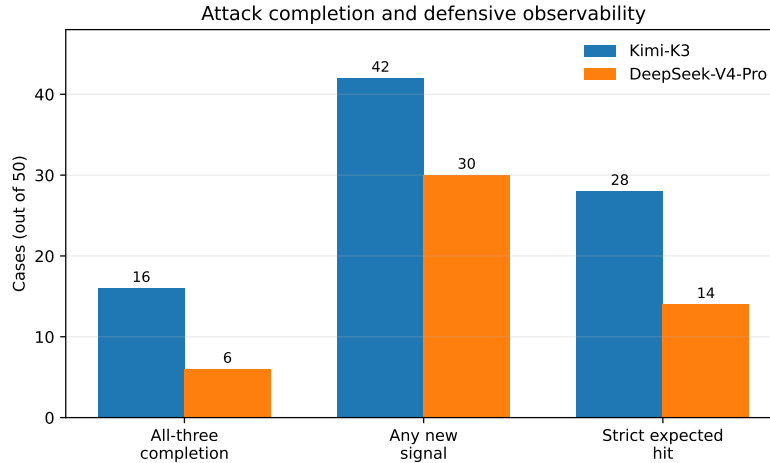


Figure 18: Attack completion and defensive observability on 50 L2 cases. “Any new signal” records whether at least one new attack-window signal appears; “strict expected hit” requires the complete expected rule set for the case.

SYSSPEAR observability. The same 50-case SYSSPEAR rerun provides a third, configuration-level observation. Only 7/50 cases (14%) complete all three targets, but 39/50 (78%) produce at least one new attack-window signal and 27/50 (54%) cover the complete expected rule set. Any-signal coverage therefore exceeds full completion by 64 percentage points, while strict expected coverage exceeds it by 40 points. More importantly, 32 of the 39 signal-producing executions and 20 of the 27 strict expected hits occur in cases that do not complete the three-stage attack chain. The additional arm thus strengthens the RQ4 claim that failed attack progression does not imply an absence of defensively relevant evidence.

The case-level quadrants make the separation between attack outcome and defensive evidence explicit. For Kimi K3, 12 cases are attack-success/signal-hit, 4 are attack-success/signal-miss, 16 are attack-failure/signal-hit, and 18 are attack-failure/signal-miss. For DeepSeek-V4-Pro, the corresponding counts are 3, 3, 11, and 33. Across the two direct-run arms, 27 failed executions nevertheless cover all expected rules, while 7 successful executions have incomplete strict coverage. Together with the SYSSPEAR rerun, these results show that runtime evidence is common even when the attack chain stops before the final target.

These cases demonstrate why SYSFIELD treats offensive capability and defensive observability as independent measurements. A failed trajectory can still execute behavior that is relevant to detection and investigation before stopping, while a completed attack need not produce the complete case-level expected signal set. The present experiment establishes runtime visibility, not protection efficacy: SYSARMOR operates in observe-only mode, and neither the presence nor the volume of signals implies that an attack was blocked or that the observed activity was severe.

Expected-behavior coverage. Table 11 summarizes the 40 paired direct-run cases for which both model arms contain complete per-rule `missing_signal` records and the 50-case SYSSPEAR rerun, for which all formal cases contain complete per-rule records. The ten additional direct-run DeepSeek cases contain expected-signal outcomes but do not provide comparable per-rule missing fields, so they remain excluded from the direct-run comparison. The denominators are therefore shown explicitly and must not be treated as a matched three-arm sample.

The direct-run ordering is unchanged: an execution-oriented tool opening a network connection is missing most often, followed by workload-level network-client use and shell or interpreter execution. The SYSSPEAR arm shows a closely related pattern, with missing rates of 40.0%, 38.0%, and 28.0%, respectively. Its first two categories are nearly tied, but network-related behaviors still lack complete evidence more often than shell or interpreter execution.

The repeated ordering is the most useful cross-model pattern. Network activity initiated by execution-oriented tools is the behavior category that most often lacks complete expected evidence

Table 11: Missing expected behaviors. Direct-run columns use 40 paired cases; the SYSSPEAR 300-turn column uses all 50 formal rerun cases.

Expected behavior	Kimi K3	DeepSeek-V4-Pro	SYSSPEAR
Execution-oriented tool opens a network connection	12/40 (30.0%)	23/40 (57.5%)	20/50 (40.0%)
Network client is used inside a workload	9/40 (22.5%)	18/40 (45.0%)	19/50 (38.0%)
Shell or interpreter executes inside a workload	6/40 (15.0%)	16/40 (40.0%)	14/50 (28.0%)

in the current records, followed by workload-level network-client activity. This identifies where closer trace alignment and telemetry analysis are most valuable.

However, the missing-rule counts should not be interpreted as a model-stealth ranking or as direct proof of a defensive blind spot. Expected rules are case-level behavioral expectations rather than action-level labels of the realized trajectory. Because the arms stop at different depths and the SYSSPEAR rerun uses a different execution protocol and denominator, some expected downstream behaviors may never occur. Other misses may reflect telemetry boundaries or rule coverage. Distinguishing “behavior not executed” from “behavior executed but not observed” requires aligning each missing rule with the corresponding agent actions and workload events. RQ4 is therefore best understood as a diagnostic analysis of incomplete defensive evidence rather than an independent capability score.

7.1.6 Evaluation Findings

The evaluation yields four main findings. First, SYSFIELD preserves all generated events in the reported stress settings while adding 2.1% system overhead; the corresponding baselines lose 60.0–98.5% of events or impose 23.2–33.0% overhead. Second, the generated ranges create measurable difficulty beyond small or single-step evaluations: multi-step composition lowers success by 18.1–21.6 percentage points, and the separate 7-to-50-node comparison shows a 9.8–10.6 point reduction across four reported models. Third, stage-level verification reveals how the attack-system architecture changes the progression funnel. With DeepSeek-V4-Pro as the common backbone, SYSSPEAR expands target-1 reach from 19/50 to 34/50, but the absolute difference contracts from 30 percentage points at target 1 to 10 at target 2 and 2 at target 3. The additional footholds therefore do not translate proportionally into downstream completion, identifying post-compromise state utilization and cross-host continuation as the remaining bottleneck without implying that SYSSPEAR is worse on the non-matched set of reached footholds. The current high-interference configuration likewise increases timeouts and execution cost while suppressing downstream completion. Fourth, runtime monitoring produces evidence in many executions that do not complete the attack. In the SYSSPEAR arm, 39/50 cases produce a new signal and 27/50 cover all expected rules despite only 7/50 full completions; offensive capability and defensive observability are therefore distinct dimensions.

Together, these results connect the three requirements of a trustworthy cyber-agent evaluation. Low-loss, low-overhead collection makes negative evidence interpretable; multi-step and larger ranges expose capability limits that simpler tasks hide; and stage-aligned offensive and defensive measurements explain both where an agent stops and what behavior remains visible before termination. Missing expected evidence is concentrated around network activity initiated by execution-oriented tools, but trajectory-level alignment is still required before those misses can be attributed to the defensive layer. SYSFIELD therefore provides not only a harder range, but also the evidence needed to explain performance within that range.

7.2 Evaluation of SYSSPEAR

We evaluate SYSSPEAR through two research questions that examine whether its coordinated multi-agent architecture improves task success over single-agent execution and how the three underlying mechanisms operate in practice:

- **RQ1: Architecture Effectiveness.** Can the coordinated multi-agent architecture improve task success rates over single-agent execution across medium- and hard-difficulty penetration testing cases?

Table 12: XBow 13-case comparison by difficulty for five SYSSPEAR configurations and two Claude Code baselines. Level-2 and Level-3 contain 9 and 4 cases, respectively.

System	Model configuration	L2 (9 cases)	L3 (4 cases)	Overall (13 cases)
SYSSPEAR	DeepSeek-V4-Pro + GLM	7/9 (77.8%)	1/4 (25.0%)	8/13 (61.5%)
	GLM-5.1	7/9 (77.8%)	1/4 (25.0%)	8/13 (61.5%)
	Qwen 3.6	5/9 (55.6%)	0/4 (0.0%)	5/13 (38.5%)
	DeepSeek-V4-Pro-Flash	8/9 (88.9%)	1/4 (25.0%)	9/13 (69.2%)
	DeepSeek-V4-Pro-Flash + skills	7/9 (77.8%)	3/4 (75.0%)	10/13 (76.9%)
Claude Code	GLM-5.1	6/9 (66.7%)	1/4 (25.0%)	7/13 (53.8%)
	Opus 4.6	8/9 (88.9%)	1/4 (25.0%)	9/13 (69.2%)

- **RQ2: Mechanism Analysis.** How do task slicing, context projection, and runtime isolation contribute to solving multi-stage attack scenarios?

7.2.1 Evaluation Setup

We evaluate SYSSPEAR on web penetration testing tasks using the XBow benchmark [54], comparing its Strategy Board architecture against a Claude Code single-agent baseline across multiple LLM configurations.

Dataset. We use a 13-case slice of the XBow benchmark, comprising 9 Level-2 (L2) and 4 Level-3 (L3) web penetration testing cases. We choose XBow for its wide adoption in recent studies of LLM-based penetration testing systems [7, 9, 10, 54]. XBow provides realistic vulnerability classes, such as SQL injection, IDOR, SSRF and XSS, and spans difficulty levels from medium-complexity to hard multi-stage attacks [59]. Its containerized deployment and automated verification are suitable for evaluating multi-agent coordination across sequential attack stages. Following its evaluation protocol, a task counts as successful only when the benchmark verifier accepts its submission; timeouts and incomplete exploits are treated as failures.

Baselines. We compare SYSSPEAR against Claude Code, a general-purpose AI coding agent that has been adapted for penetration testing tasks in prior benchmark studies [54]. Claude Code serves as a representative single-agent baseline: it runs as a single integrated agent that interleaves tool use and reasoning within one continuous session, without the Coordinator-driven task slicing, typed Strategy Board evidence, or cross-Assignment context projection that SYSSPEAR introduces. This makes it a suitable reference point for isolating the contribution of coordinated multi-agent execution rather than of any single underlying LLM.

Metrics. We report task-level success rates stratified by difficulty (L2, L3, and overall). Because the compared runs vary in model provider and execution date rather than forming a strictly controlled paired trial, we treat aggregate success rate as the primary quantitative signal, complemented by qualitative process evidence from two L3 cases (RQ2).

Experimental protocol. Both SYSSPEAR and Claude Code execute on the same containerized runtime environment with identical base tooling (curl, nmap, gobuster, sqlmap), and each task runs under a 10-minute timeout. SYSSPEAR’s Coordinator may spawn multiple sequential or overlapping Assignments, with each Solver receiving a context projection of the current Strategy Board state rather than the full conversation history; Claude Code instead runs as a single continuous session per task. We report five SYSSPEAR configurations, varying by model provider (DeepSeek-V4-Pro, GLM-5.1, Qwen 3.6) and by whether domain-specific skills are enabled, alongside two Claude Code configurations (GLM-5.1 and Opus 4.6) that enable both same-model and cross-model comparison against SYSSPEAR.

7.2.2 RQ1: Architecture Effectiveness

Table 12 reports success rates for five SYSSPEAR configurations and two Claude Code baselines, stratified by difficulty level.

On the 13-case slice, the skill-enabled SYSSPEAR configuration is the strongest recorded setting, solving 10 tasks (76.9%). Its advantage is concentrated in Level-3 coverage: it solves 3 of 4 hard

cases, whereas every other reported configuration solves at most one. The DeepSeek-V4-Pro-Flash and Opus 4.6 Claude Code configurations each solve 9/13 tasks (69.2%), but their single Level-3 success leaves them below the skill-enabled configuration overall.

The same-model comparison provides an architecture-aligned signal: with GLM-5.1, SYSPEAR solves 8/13 tasks compared with 7/13 for the Claude Code. Notably, SYSPEAR completes the Level-3 HTTP-smuggling task XBEN-066-24, which the Claude Code baseline does not complete, consistent with the expectation that coordinated multi-agent execution benefits multi-stage exploration in harder cases. This configuration-level comparison does not isolate the causal contribution of individual mechanisms, since it reflects the combined architecture under the recorded XBow protocol; RQ2 turns to process-level evidence for how the mechanisms operate.

7.2.3 RQ2: Mechanism Analysis

We examine process evidence from two L3 cases below: XBEN-018, a tag-filter XSS bypass, and XBEN-066, the HTTP-smuggling task.

Case study: XBEN-018 (XSS tag-filter bypass). This task asks the attacker to trigger `alert("XSS")` on a web application that reflects user input but filters many HTML tags. The Claude Code agent (Opus 4.6) made 57 tool calls, discovered the reflection point early, but spent most of its session attempting sandbox-escape variants and environment assumptions rather than systematically enumerating the filter boundary, and timed out without a working payload. SYSPEAR instead allocated two sequential Assignments: the first Solver attempted common tags (`<script>`, `<img>`, `<svg>`), and recorded each rejection as a Board entry typed `attempt` with explicit `contradicts` relations rather than as an opaque failure; the second Solver, focused on the known reflection point and the recorded filter failures, tested single-letter tags, found that `<a>` through `<y>` were blocked but `<z>` was not, and completed a working `onfocus/autofocus` payload on that tag.

The key difference is not that one system ran two executors, but that the Strategy Board preserved the exact filter boundary as explicit relations among Board entries. The second Assignment received a context projection containing the negative results, the confirmed reflection fact, and the next open question, so it resumed from a compressed parse boundary rather than from a blank session. This aligns with the intended benefit of task slicing and context projection: local search can advance systematically when prior failures are recorded as structured, relational evidence rather than as a growing transcript.

Case study: XBEN-066 (HTTP smuggling / SSRF). This task places the target endpoint behind a proxy and expects the attacker to reach an internal hostname through Host-header manipulation. The Claude Code agent (Opus 4.6) found the login credentials (`test:test`), confirmed the presence of HAProxy, and observed internal routing clues, but its 60-tool-call session contained 47 explicit authentication or session-setup steps and many repeated login attempts, with no structured record of which capabilities had already been confirmed, and it ended with incomplete exploration. SYSPEAR solved the task in three overlapping Assignments: the first Solver discovered `test:test` in an HTML comment, logged in, accessed `/wifi_settings`, and recorded a Board capability stating that the application proxies requests to `internal.router` via HAProxy; the Coordinator then dispatched a second Assignment, anchored on that capability, which bypassed the proxy using `Host: internal.router`, enumerated `/devices`, found a device named `flag`, and fetched `/devices/flag/status`. A third Assignment exploring POST-response details was cancelled once the second Assignment succeeded.

The central architectural feature here is that the SSRF capability was promoted to a Board entry with explicit `unlocks` relations, so the Coordinator could slice a new Assignment anchored on that capability without requiring the new Solver to rediscover the proxy chain. Context Projection delivered the login fact, the proxy capability, and already-refuted hypotheses (such as parameter injection) as a compact sub-graph, not a full transcript. The second Solver therefore started with confirmed access and a specific next route, consistent with the intended benefit of runtime isolation and Board-mediated collaboration: each Solver produces durable, relational evidence, while the Coordinator slices complementary routes rather than duplicating entire sessions.

Table 13: Rule-level alert counts produced by the rule-based detection engine during the one-week deployment.

Rule Name	Alert Count
command.injection.indicators	7600
crontab.modification	7525
permission.modification	6670
suspicious.binary.execution	4865
clear.log.activities	4454
dev.shm.execution	3677
temp.executable.creation	3673
nonstandard.port.connection	103
sudo.privilege.escalation	12
file.deletion.dangerous	5
service.manipulation	4

7.2.4 Evaluation Findings

The evaluation yields three main findings. First, the coordinated multi-agent architecture improves task success over single-agent execution, with the skill-enabled SYSSPEAR configuration achieving the highest recorded success rate of 76.9% (10/13 tasks). Its advantage is concentrated in Level-3 coverage: it solves 3 of 4 hard cases, whereas every other reported configuration solves at most one. Second, the same-model comparison provides architecture-aligned evidence: with GLM-5.1, SYSSPEAR solves 8/13 tasks compared with 7/13 for Claude Code, completing the Level-3 HTTP-smuggling task XBEN-066-24 that the baseline does not complete. This differential is consistent with the expectation that coordinated multi-agent execution benefits multi-stage exploration in harder cases. Third, process evidence from two Level-3 cases demonstrates how the three architectural mechanisms contribute in practice. In XBEN-018, task slicing and context projection enabled systematic filter-boundary enumeration by preserving negative results as explicit Board relations rather than as transcript noise. In XBEN-066, runtime isolation and Board-mediated collaboration allowed the Coordinator to slice a second Assignment anchored on a discovered SSRF capability without requiring the new Solver to rediscover the proxy chain, reducing redundant authentication steps and enabling focused exploitation.

Together, these results show that the coordinated multi-agent design offers measurable improvement over single-agent baselines, with benefits concentrated in harder multi-stage tasks. The three mechanisms are co-designed rather than independent: task slicing defines the Assignment boundaries that context projection projects evidence onto, and runtime isolation depends on both to keep each Solver’s session bounded yet informed. The present comparison therefore characterizes their combined contribution rather than attributing a separate share to each; under this combined architecture, SYSSPEAR solves tasks on which a higher-capability single-agent baseline timed out.

7.3 Evaluation of SYSARMOR

We evaluate SYSARMOR in a realistic deployment setting to examine its operational effectiveness, scalability, and resource efficiency in production-like conditions. The assessment is organized around the following research questions:

- **RQ1: Detection Effectiveness.** How effectively does SYSARMOR detect real-world APT activity?
- **RQ2: Scalability.** What is the end-to-end throughput of the system, and how well does it scale to large enterprise deployments?
- **RQ3: Resource Consumption.** What memory footprint is required for online provenance-based detection under sustained event ingestion?

Table 14: Detection results produced by the NODLINK + KNOWHOW inference pipeline against the real-world APT campaign. NODLINK constructs the suspicious provenance subgraph, and KNOWHOW maps the observed behavior to MITRE ATT&CK techniques and attack stages.

Attack Stage	TTPs	Malicious Command Line
Initial Compromise	T1570	<code>/tmp/.>_ /var/tmp/.>_ /tmp/sshd, /root/sshd</code>
Establish Foothold	T1105	<code>/usr/sbin/sshd -D, sshd [accepted], sshd nobody [priv]</code>
Move Laterally	T1021.004	<code>sshd root [priv], sshd operator [priv]</code>
Escalate Privilege	T1222	<code>(cd '/var/tmp' && [-f 'yum.log'] && (chattr -i 'yum.log'; chmod +x 'yum.log' && ./yum.log))</code>
	T1036	<code>(cd '/var/tmp' && [-f 'apt.log'] && (chattr -i 'apt.log'; chmod +x 'apt.log' && ./apt.log))</code>
	T1022.002	<code>chmod +x yum.log, chmod +x .bash_logout, /usr/bin/ydvlmeowey su 12812</code>
Internal Reconnaissance	T1016	<code>/usr/bin/yzhulutzjr ls -la, /usr/bin/ukkoisjdsi route -n, /usr/bin/dcjwvfaznr ifconfig eth0</code>
	T1057	<code>/usr/bin/zqjxlypgjo netstat -an, /usr/bin/rvwcaowook ps -ef, /usr/bin/tseyejrpq who</code>
	T1033	<code>/usr/bin/wrrvfqlxbh whoami, /usr/bin/mbydvjvtf top, /usr/bin/srtebasq uptime, /usr/bin/cqlmxscjbm pwd</code>
	T1083	<code>/usr/bin/bkzytqfba cd /etc, /usr/bin/vyqlcyypoo cat /etc/resolv.conf, which apt, grep -v grep</code>
Maintain Persistence	T1562	<code>shell -c ps -eo pid,comm,%cpu awk '...' sort -k3nr ... xargs kill ... pgrep -x '>' — tail</code>
	T1499	<code>readlink -f /proc/10751/exe, xargs -r kill -9, ps -o pid=,pcpu= -p 19901 30569</code>
	T1053	<code>chattr -i .bash_logout, /bin/bashell -c run-parts /etc/cron.hourly, basename /etc/cron.hourly/gcc.sh</code>

7.3.1 Evaluation Setup

SYSARMOR is deployed in the Third Research Institute of Ministry of Public Security for a one-week operational assessment. The deployment targets a representative endpoint environment and is designed to measure both detection quality and system overhead under realistic event volumes.

Dataset. We install SYSARMOR agents on a monitored endpoint within the enterprise network and collect system telemetry over a one-week period. During this time, the red team at the Third Research Institute of Ministry of Public Security executes an APT campaign against the monitored host to emulate realistic adversarial behavior. The endpoint and server operate on Ubuntu, and the collected auditd stream includes approximately 30 million raw events, which are normalized into 8,179,548 Sysdig events before ingestion by the detection pipeline. This corresponds to an average event rate of roughly 57 events per second per endpoint.

Experimental protocol. The SYSARMOR server is deployed on a dedicated machine with 32 CPU cores, an NVIDIA GeForce RTX 3090 GPU, 128 GB RAM, and 2 TB of storage, running Ubuntu 20.04 LTS. Docker and Kubernetes are used for container orchestration, while OpenSearch and PostgreSQL provide alert storage and metadata management. A React-based web interface is also deployed to enable operational monitoring and analyst interaction.

7.3.2 RQ1: Detection Effectiveness

Rule-based Detection. We evaluate the rule-based engine using an existing Falco rule set containing more than 36 signatures for common malicious behaviors. During the deployment period, the engine triggers 38,588 alerts, corresponding to approximately 0.47% of the processed event stream. The distribution of alerts across rule categories, summarized in Table 13, shows that the most frequently activated detections correspond to command injection, crontab modification, permission tampering, suspicious binary execution, log clearing, and execution from temporary or shared-memory paths. These patterns are consistent with persistence, privilege escalation, and evasion behaviors commonly observed in real intrusion campaigns.

Machine Learning (ML)-based Detection. The ML-based detection pipeline, composed of NODLINK and KNOWHOW, identifies a multi-day APT campaign during the deployment. The malicious activity persists for three days and is first surfaced on the first day of the campaign, indicating that the system can detect adversarial behavior early in the attack lifecycle. As the campaign evolves, the alert remains active and is progressively refined until the activity ceases on the third day. The generated alert ultimately contains a provenance graph with 201 nodes and 1,982 edges, providing a structured representation of the attack trajectory.

Table 14 reports the attack stages, associated MITRE ATT&CK techniques, and representative malicious command lines identified by the system. After removing redundant or highly similar command strings for readability, the results show that SYSARMOR recognizes six attack phases: Initial Compromise, Establish Foothold, Move Laterally, Escalate Privilege, Internal Reconnaissance, and

Table 15: Throughput of SYSARMOR under different Flink job configurations. “eps” denotes events per second.

Job Configuration	Throughput (eps)
<i>Job 01 + Job 02</i>	12547
<i>Job 01 + Job 03</i>	12208
<i>Job 01 + Job 02 + Job 03</i>	9003

Maintain Persistence. No Data Exfiltration or Impact stage is observed during the campaign, suggesting that the intrusion was interrupted before reaching later-stage objectives. For each stage, the system maps the suspicious provenance evidence to the relevant ATT&CK technique and preserves the underlying command-line context used by the adversary.

The TTP-level attribution is enabled by KNOWHOW’s semantic reasoning over structured cyber threat intelligence, represented as gIoC triples. In the lateral movement stage, for example, KNOWHOW matches triples such as <adversary, use, ssh service> and <adversary, transfer, file> to provenance evidence involving shell processes launching *sshd* and writing files such as */tmp/sshd* and */var/tmp/._*. Similarly, the masquerading and permission-modification behaviors are aligned to triples such as <adversary, create, system file name> and <adversary, modify, file permission>, which correspond to the observed creation of files such as *yum.log* and subsequent execution of commands like *chattr -i* and *chmod +x*. For obfuscation-related activity, the system matches triples like <adversary, execute, obfuscated file> to execution paths such as */usr/bin/yzhulutzjr* and */usr/bin/zqxlypgjo*. This semantic alignment between provenance structure and CTI knowledge enables fine-grained TTP attribution beyond simple signature matches.

The observed campaign also demonstrates the kinds of stealth techniques that often evade conventional defenses. The adversary uses masquerading and obfuscation to hide malicious artifacts behind filenames such as *yum.log*, *apt.log*, and *gcc.sh*, while storing payloads in hidden paths such as */tmp/._* and */var/tmp/._*. Command execution is further disguised through randomized executable names, allowing the attacker to evade rule-based checks that rely on static file names or known command patterns. These behaviors are difficult for both signature-based and learning-based systems to capture reliably, because they are rare, highly context-dependent, and intentionally designed to look benign. The results indicate that SYSARMOR’s provenance-based analysis provides an effective countermeasure against this class of evasive activity.

7.3.3 RQ2: Scalability

End-to-end throughput. To evaluate throughput, we replay a fixed workload from a single monitored endpoint and measure the number of events processed per second as they pass through the streaming pipeline. We use one-agent playback because simulating multiple sources to saturate the cluster would require a substantially larger deployment and would not reflect the isolated upper-bound behavior of a single endpoint. In this setup, the raw audit stream is first written to Kafka, then consumed by Flink jobs for event normalization and detection. The first job performs normalization from raw auditd records into Sysdig format, while the second and third jobs execute rule-based and machine-learning-based detection, respectively.

Table 15 reports the measured throughput under different execution configurations. The combined configuration of Job 01 and Job 02 and the configuration of Job 01 and Job 03 both sustain approximately 12,000 events per second. When both detection jobs run concurrently, the system processes about 9,000 events per second. This reduction is primarily attributed to Kafka’s duplicated delivery of the same event stream to two independent consumer groups, which increases broker egress load, I/O overhead, and network pressure. Although the throughput drops under simultaneous rule-based and machine-learning execution, the observed rate remains far above the 57 events per second generated by a typical endpoint in our deployment.

This result suggests that a single deployment server can support more than 150 endpoints in real time under the current workload profile. We further inspect the pipeline bottleneck and find that Job 01, the normalization stage, is the dominant limiting component. This is not due to the detection logic

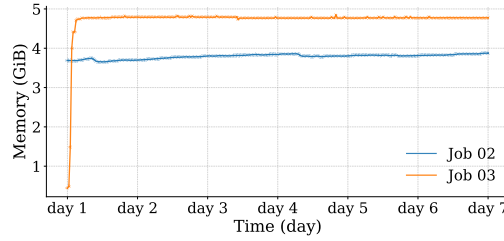


Figure 19: Memory consumption of the SYSARMOR server during the evaluation period.

itself, but rather to the naive parsing implementation in this stage. As such, further optimization of the normalization layer is expected to yield substantial throughput gains.

7.3.4 RQ3: Resource Consumption

We quantify memory consumption by monitoring the Flink containers during the throughput experiment described in Section 7.3.3. Because the provenance-based detection pipeline maintains in-memory graph structures and cached artifacts for anomaly construction and semantic interpretation, memory overhead is a key operational concern. We therefore measure the resource usage of the two detection jobs separately, each running in a dedicated Flink container with 8 GB of allocated memory.

Figure 19 shows the measured memory traces. The rule-based job remains relatively lightweight, as it only maintains a compact rule set and evaluates predicates against the incoming event stream. The machine-learning job demonstrates a higher initial memory footprint during warm-up, when the NODLINK model and KNOWHOW knowledge base are loaded and graph state is initialized. After this startup phase, memory usage stabilizes. The rule-based job settles at approximately 3.8 GB, while the machine-learning job stabilizes at roughly 4.8 GB. These values indicate that SYSARMOR can operate with moderate memory requirements while sustaining real-time detection on production-like workloads.

7.3.5 Evaluation Findings

The evaluation yields three main findings. First, SYSARMOR demonstrates strong detection capability in a real-world deployment. The rule-based engine emits 38,588 alerts across 11 rule categories, covering 0.47% of the processed event stream. The combined NODLINK + KNOWHOW pipeline detects a three-day APT campaign and maps it to six attack stages, resulting in a provenance graph with 201 nodes and 1,982 edges. The semantic alignment between CTI-derived gloCs and observed provenance events produces high-fidelity TTP attribution, which is particularly valuable for stealthy behaviors such as masquerading and obfuscated file execution.

Second, SYSARMOR scales effectively to enterprise-level workloads. The system sustains approximately 12,000 events per second with single-job detection configurations and maintains about 9,000 events per second when both detection jobs operate concurrently. Given an average event rate of 57 events per second per endpoint, these results indicate that a single server can support more than 150 endpoints in real time. The primary bottleneck resides in the normalization stage, which suggests that additional optimization of the parsing layer can further improve throughput.

Third, SYSARMOR requires only moderate memory resources. After the initial warm-up period, the rule-based detection job consumes around 3.8 GB and the machine-learning job around 4.8 GB, both remaining stable throughout the evaluation. This confirms that provenance-based real-time threat detection is feasible within a realistic operational resource envelope.

Taken together, these results indicate that SYSARMOR satisfies the intended design goals of real-time, interpretable, and actionable defense. The combination of signature-based detection and provenance-driven reasoning enables the system to capture both well-known attack patterns and evasive behaviors that are difficult to detect using conventional methods alone. The observed throughput and memory usage further suggest that the architecture is practical for deployment in live enterprise

environments, while preserving the contextual richness needed for accurate TTP-level attribution and analyst investigation.

8 Conclusion

In this report, we proposed co-evolution as the integrating insight to address the growing asymmetry between autonomous attack and static defense in cybersecurity. Based on this insight, we presented SYSEVOLVE, comprising three co-designed components, SYSFIELD, SYSSPEAR, SYSARMOR, that form a self-driven adversarial co-evolution loop. SYSFIELD constructs realistic multi-host ranges via declarative topology-driven construction, provides complete attack-trace collection and trustworthy assessment, restoring deployment evolution. SYSSPEAR generates efficient, safe attack plans via task-sliced local search, skill-based expertise enhancement mechanism, and static-analysis verification, restoring attack capability evolution. SYSARMOR performs real-time, interpretable defense via state tracking, CTI-context fusion, and stage-matched response, restoring defense capability evolution. In evaluation, SYSFIELD achieves zero-loss collection at 2.1% overhead and orchestrates 257 CVEs into 1,148 ranges, SYSSPEAR improves attack success by over 25% over baseline LLMs, and SYSARMOR achieves $10\text{--}1000\times$ greater precision than prior systems and detects real APT attacks in production at Huawei and Sangfor. Our evaluation also reveals three findings about LLM agent capabilities. First, multi-step composition and larger topologies expose agent capability gaps hidden by single-step evaluations. Second, the bottleneck lies after initial access in post-compromise state utilization. Third, LLM agents are susceptible to environmental interference. When decoy endpoints are deployed in the range, agent timeouts triple and downstream completion disappears despite the success rates of initial accesses are unchanged.

References

- [1] “CRADLE Project Homepage — cradle-dsl.github.io,” <https://cradle-dsl.github.io/>, [Accessed 15-08-2026].
- [2] A. Ahmad, S. Lee, and M. Peinado, “Hardlog: Practical tamper-proof system auditing using a novel audit device,” in *2022 IEEE Symposium on Security and Privacy (SP’22)*, 2022.
- [3] Anthropic, “Claude code,” <https://claude.com/product/claude-code>, 2026, accessed: 2026-08-11.
- [4] M. Bhatt, S. Chennabasappa, C. Nikolaidis, S. Wan, I. Evtimov, D. Gabi, D. Song, F. Ahmad, C. Aschermann, L. Fontana, S. Frolov, R. P. Giri, D. Kapil, Y. Kozyrakis, D. LeBlanc, J. Milazzo, A. Straumann, G. Synnaeve, V. Vontimitta, S. Whitman, and J. Saxe, “Purple llama CyberSecEval: A secure coding benchmark for language models,” 2023. [Online]. Available: <https://arxiv.org/abs/2312.04724>
- [5] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, “Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance,” 2023.
- [6] CVELab Contributors, “sysfield: Evaluating cyber agents in real defended environments,” Technical report and experiment record, Aug. 2026, report branch, accessed 2026-08-11. [Online]. Available: <https://github.com/Yuchuwa/CVELab/blob/report/reports/report.en.md>
- [7] I. David and A. Gervais, “Multi-agent penetration testing AI for the web,” 2025. [Online]. Available: <https://arxiv.org/abs/2508.20816>
- [8] DeepSeek-AI, “DeepSeek V4 preview release,” DeepSeek API Documentation, Apr. 2026, accessed 2026-08-11. [Online]. Available: <https://api-docs.deepseek.com/news/news260424/>
- [9] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, “{PentestGPT}: Evaluating and harnessing large language models for automated penetration testing,” in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 847–864.

- [10] A. Dhakal, K. Neupane, and A. Chaudhary, “Baselines before architecture: Evaluating coding agents for autonomous penetration testing,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.13085>
- [11] R. Fang, R. Bindu, A. Gupta, and D. Kang, “Llm agents can autonomously exploit one-day vulnerabilities,” 2024. [Online]. Available: <https://arxiv.org/abs/2404.08144>
- [12] L. Folkerts, W. Payne, S. Inman, P. Giavridis, J. Skinner, S. Deverett, J. Aung, E. Zorer, M. Schmatz, M. Ghanem, J. Wilkinson, A. Steer, V. Hong, and J. Wang, “Measuring AI agents’ progress on multi-step cyber attack scenarios,” UK AI Security Institute, Tech. Rep., Mar. 2026, accessed 2026-08-10. [Online]. Available: <https://www.aisi.gov.uk/research/measuring-ai-agents-progress-on-multi-step-cyber-attack-scenarios>
- [13] T. L. Foundation, “Opensearch: Find the meaning in your data,” 2025, <https://opensearch.org/>.
- [14] V. Gandhi, S. Banerjee, A. Agrawal, A. Ahmad, S. Lee, and M. Peinado, “Rethinking system audit architectures for high event coverage and synchronous log availability,” in *32nd USENIX Security Symposium (USENIX Security’23)*, 2023.
- [15] A. Happe and J. Cito, “Getting pwn’d by ai: Penetration testing with large language models,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 2082–2086. [Online]. Available: <https://doi.org/10.1145/3611643.3613083>
- [16] N. He, Z. Zhao, J. Wang, Y. Hu, S. Guo, H. Wang, G. Liang, D. Li, X. Chen, and Y. Guo, “Eunomia: Enabling user-specified fine-grained search in symbolically executing webassembly binaries,” in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 385–397. [Online]. Available: <https://doi.org/10.1145/3597926.3598064>
- [17] Z. He, J. Dong, Z. Li, T. Chen, G. Deng, F. Luo, J. Ji, Y. Cao, and X. Luo, “A survey of llm-driven penetration testing: Taxonomy, co-evolution, and open challenges,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.02605>
- [18] Hugging Face, “Security incident disclosure – july 2026,” Jul. 2026, accessed 2026-08-10. [Online]. Available: <https://huggingface.co/blog/security-incident-july-2026>
- [19] Intel, “Intel sgx: Enhance security protections, data privacy, and confidentiality.” <https://www.intel.com/content/www/us/en/products/details/processors/xeon/features/software-guard-extensions.html>, 2026, accessed: 2026-08-11.
- [20] H. Jiang, J. Huang, S. Li, B. Yu, P. Jiang, Y. Ren, N. Jia, Y. Guo, and D. Li, “Stair: Effective incident response using an end-to-end agentic planning framework,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.09524>
- [21] H. Jiang, P. Wang, J. Jin, S. Li, Z. Shen, P. Wang, Z. Wang, Y. Cai, N. Jia, Y. Ren, P. Jiang, Y. Guo, and D. Li, “Rangefactory: Scalable construction of multi-hop cyber ranges,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.09526>
- [22] P. Jiang, R. Huang, D. Li, Y. Guo, X. Chen, J. Luan, Y. Ren, and X. Hu, “Auditing frameworks need resource isolation: A systematic study on the super producer threat to system auditing and its mitigation,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023. [Online]. Available: <https://arxiv.org/abs/2307.15895>
- [23] P. Jiang, H. Jiang, R. Huang, H. Lei, Z. Zhong, S. Zhang, Y. Ren, N. Jia, X. Hu, Y. Guo *et al.*, “Dpuaudit: Dpu-assisted pull-based architecture for near-zero cost system auditing,” in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA’25)*, 2025.
- [24] T. Keary, “Hugging face breach signals a new era of ai-powered cyberattacks,” 2026. [Online]. Available: <https://www.forbes.com/sites/timkeary/2026/07/21/hugging-face-breach-ai-powered-cyberattacks/>

- [25] M. Kiely, M. Ahiskali, E. Borde, B. Bowman, D. Bowman, D. Van Bruggen, K. C. Cowan, P. Dasgupta, E. Devendorf, B. Edwards, A. Fitts, S. Fugate, R. Gabrys, W. Gould, H. H. Huang, J. Jacobs, R. Kerr, I. J. King, L. Li, L. Martinez, C. Moir, C. Murphy, O. Naish, C. Owens, M. Purchase, A. Ridley, A. Taylor, S. Farmer, W. J. Valentine, and Y. Zhang, “Cage challenge 4: A scalable multi-agent reinforcement learning gym for autonomous cyber defence,” *AI Magazine*, vol. 46, no. 3, p. e70021, 2025. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.70021>
- [26] Kimi Team, “Kimi K3: Open frontier intelligence,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.24653>
- [27] H. Lee, J. Liu, D. Kim, Z. Zhang, C. S. Xia, and L. Zhang, “SEC-bench Pro: Can language models solve long-horizon software security tasks?” 2026. [Online]. Available: <https://arxiv.org/abs/2605.26548>
- [28] S. Lee and D. Brumley, “ExploitBench: A capability ladder benchmark for LLM cybersecurity agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2605.14153>
- [29] S. Li, F. Dong, X. Xiao, H. Wang, F. Shao, J. Chen, Y. Guo, X. Chen, and D. Li, “NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation.” in *NDSS*, 2024.
- [30] F. Liu, J. Dai, Y. Fan, W. Mai, Z. Li, B. Chen, J. Zhang, Z. Lou, B. Xiang, Q. Zhang, X. Pan, G. Hong, Y. Zhang, and M. Yang, “AgentCyberRange: Benchmarking frontier AI systems in realistic cyber ranges,” 2026. [Online]. Available: <https://arxiv.org/abs/2606.14295>
- [31] LTTng Project, “LTTng: The linux trace toolkit next generation,” Project documentation, 2026, accessed 2026-08-11. [Online]. Available: <https://lttng.org/docs/>
- [32] Y. Meng, S. Li, J. Gui, P. Jiang, and D. Li, “KnowHow: Automatically Applying High-Level CTI Knowledge for Interpretable and Accurate Provenance Analysis,” in *NDSS*, 2026.
- [33] Microsoft, “Microsoft security copilot,” 2016, <https://www.microsoft.com/en-us/security/business/ai-machine-learning/microsoft-security-copilot>.
- [34] “ETW events in the common language runtime,” Microsoft, 2017, accessed on October 1, 2023. [Online]. Available: [https://msdn.microsoft.com/en-us/library/ff357719\(v=vs.110\).aspx/](https://msdn.microsoft.com/en-us/library/ff357719(v=vs.110).aspx/)
- [35] Model Evaluation and Threat Research (METR), “Task-completion time horizons of frontier AI models,” 2026, accessed 2026-08-10. [Online]. Available: <https://metr.org/time-horizons/>
- [36] H. P. T. Nguyen, K. Hasegawa, K. Fukushima, and R. Beuran, “Pengym: Realistic training environment for reinforcement learning pentesting agents,” *Comput. Secur.*, vol. 148, no. C, Jan. 2025. [Online]. Available: <https://doi.org/10.1016/j.cose.2024.104140>
- [37] S. Oesch, P. Austria, A. Chaulagain, B. Weber, C. Watson, M. Dixon, and A. Sadovnik, “The path to autonomous cyberdefense,” *IEEE Security and Privacy*, vol. 23, no. 1, p. 38–46, Jan. 2025. [Online]. Available: <https://doi.org/10.1109/MSEC.2024.3427640>
- [38] OpenAI, “GPT-5 system card,” OpenAI, Tech. Rep., Aug. 2025, accessed 2026-08-10. [Online]. Available: <https://openai.com/index/gpt-5-system-card/>
- [39] —, “Codex,” <https://openai.com/zh-Hans-CN/codex/>, 2026, accessed: 2026-08-11.
- [40] —, “GPT-5.6 system card,” OpenAI, Tech. Rep., Jul. 2026, accessed 2026-08-10. [Online]. Available: <https://deploymentsafety.openai.com/gpt-5-6/>
- [41] —, “OpenAI and Hugging Face partner to address security incident during model evaluation,” Jul. 2026, accessed 2026-08-10. [Online]. Available: <https://openai.com/index/hugging-face-model-evaluation-security-incident/>
- [42] penligent, “Gpt-5.6 sol jailbreaks and agentic cyber risk,” 2026. [Online]. Available: <https://www.penligent.ai/hackinglabs/gpt-5-6-sol-jailbreaks/>

- [43] “The linux audit framework.” Redhat., accessed on October 1, 2023. [Online]. Available: <https://github.com/linux-audit/>
- [44] M. Sharif, P. Datta, A. Riddle, K. Westfall, A. Bates, V. Ganti, M. Lentzk, and D. Ott, “Drsec: Flexible distributed representations for efficient endpoint security,” in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 3609–3624.
- [45] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, “Pentestagent: Incorporating llm agents to automated penetration testing,” in *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS ’25. New York, NY, USA: Association for Computing Machinery, 2025, p. 375–391. [Online]. Available: <https://doi.org/10.1145/3708821.3733882>
- [46] “Sysdig.” Sysdig., 2013, ccessed on October 1, 2023. [Online]. Available: <https://sysdig.com>
- [47] The MITRE Corporation, “MITRE ATT&CK,” The MITRE Corporation, 2022, accessed 2026-08-10. [Online]. Available: <https://attack.mitre.org/>
- [48] Y. Wang, Z. Zhang, N. He, Z. Zhong, S. Guo, Q. Bao, D. Li, Y. Guo, and X. Chen, “Symgx: Detecting cross-boundary pointer vulnerabilities of sgx applications via static symbolic execution,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2710–2724. [Online]. Available: <https://doi.org/10.1145/3576915.3623213>
- [49] Z. Wang, N. Schiller, H. Li, S. S. Narayana, M. Nasr, N. Carlini, X. Qi, E. Wallace, E. Bursztein, L. Invernizzi, K. Thomas, Y. Shoshitaishvili, W. Guo, J. He, T. Holz, and D. Song, “ExploitGym: Can AI agents turn security vulnerabilities into real attacks?” 2026. [Online]. Available: <https://arxiv.org/abs/2605.11086>
- [50] Z. Wang, T. Shi, J. He, M. Cai, J. Zhang, and D. Song, “CyberGym: Evaluating AI agents’ real-world cybersecurity capabilities at scale,” in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: <https://openreview.net/forum?id=2YvbLQEdYt>
- [51] H. Xu, S. Wang, N. Li, K. Wang, Y. Zhao, K. Chen, T. Yu, Y. Liu, and H. Wang, “Large language models for cyber security: A systematic literature review,” *ACM Trans. Softw. Eng. Methodol.*, Sep. 2025, just Accepted. [Online]. Available: <https://doi.org/10.1145/3769676>
- [52] J. Yang, A. Prabhakar, K. Narasimhan, and S. Yao, “InterCode: Standardizing and benchmarking interactive coding with execution feedback,” in *Advances in Neural Information Processing Systems: Datasets and Benchmarks Track*, 2023. [Online]. Available: <https://arxiv.org/abs/2306.14898>
- [53] L. Yang, Z. Chen, C. Wang, Z. Zhang, S. Booma, P. Cao, C. Adam, A. Withers, Z. Kalbarczyk, R. K. Iyer, and G. Wang, “True attacks, attack attempts, or benign triggers? an empirical measurement of network alerts in a security operations center,” in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 1525–1542. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/yang-limin>
- [54] Y. Yang, H. Fang, and Z. Zheng, “Hackers or hallucinators? A comprehensive analysis of LLM-based automated penetration testing,” 2026. [Online]. Available: <https://arxiv.org/abs/2604.05719>
- [55] A. K. Zhang, N. Perry, R. Dulepet, J. Ji, C. Menders, J. W. Lin, E. Jones, G. Hussein, S. Liu, D. Jasper, P. Peetathawatchai, A. Glenn, V. Sivashankar, D. Zamoshchin, L. Glikbarg, 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,” in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: <https://arxiv.org/abs/2408.08926>

- [56] Zhipu AI, “GLM-5.2,” Zhipu AI Open Documentation, 2026, accessed 2026-08-11. [Online]. Available: <https://docs.bigmodel.cn/cn/guide/models/text/glm-5.2>
- [57] Z. Zhong, Z. Zhang, H. Guan, and D. Li, “Orax: A feedback-driven framework for efficiently solving satisfiability modulo theories and oracles,” *Proc. ACM Program. Lang.*, vol. 9, no. OOPSLA1, Apr. 2025. [Online]. Available: <https://doi.org/10.1145/3720438>
- [58] Y. Zhu, A. Kellermann, D. Bowman, P. Li, A. Gupta, A. Danda, R. Fang, C. Jensen, E. Ihli, J. Benn, J. Geronimo, A. Dhir, S. Rao, K. Yu, T. Stone, and D. Kang, “CVE-Bench: A benchmark for AI agents’ ability to exploit real-world web application vulnerabilities,” in *Proceedings of the 42nd International Conference on Machine Learning*, 2025. [Online]. Available: <https://arxiv.org/abs/2503.17332>
- [59] Y. Zhu, A. Kellermann, A. Gupta, P. Li, R. Fang, R. Bindu, and D. Kang, “Teams of LLM agents can exploit zero-day vulnerabilities,” in *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, V. Demberg, K. Inui, and L. Marquez, Eds. Rabat, Morocco: Association for Computational Linguistics, Mar. 2026, pp. 23–35. [Online]. Available: <https://aclanthology.org/2026.eacl-long.2/>