

Syscon: Kernel-Level Syscall Monitoring for AI Agent Control in Sandboxed Environments*

Jade Sheffey¹, Joyce Levine¹, Willow Taylor¹, and Joyce Werhane¹

¹University of Massachusetts Amherst

With Apart Research

Abstract

Many existing AI control protocols rely on a *trusted monitor* to evaluate an untrusted agent’s actions, but these monitors generally access tool call invocations and context, not what the agent actually does at the system level. We identify this as a fundamental information gap in the control stack and present Syscon, the first system to close it. Syscon is a supervisor-side auditor daemon that monitors agent activity at the Linux kernel syscall boundary, leveraging `seccomp-bpf` audit logging from *outside* the agent’s container to observe all process creation, network connections, and filesystem operations, all without running any code inside the container and without the agent’s knowledge. The system produces compressed, per-action *receipts* designed for consumption by a trusted monitor LLM. We evaluate Syscon on 234 attack trajectories across 9 attacker models in the LinuxArena control evaluation framework. Our qualitative analysis confirms that receipts surface behavioral signals (credential file reads, payload staging, reverse shell listeners, data exfiltration) that are invisible in tool call text, covering multiple side task categories. However, a first-pass integration with a Gemini 2.5 Flash monitor did not improve detection rates, establishing that *observability and detection are distinct research problems*: Syscon receipts carry signal, but monitors must be taught to use it.

1 Introduction

When an AI agent runs a shell command inside a container, the trusted monitor sees the command text, and occasionally some context. The kernel sees hundreds of system calls (process forks, network connections, file operations, IPC communication), any of which could be part of an attack. This paper gives the monitor access to the kernel’s view without introducing undue overhead to the evaluation environment.

AI companies are beginning to deploy autonomous agents to automate complex software engineering and system administration tasks [9]. These agents operate by issuing shell commands on target hosts, where they require broad permissions (root access, network connectivity, package installation) to accomplish their work. As agent capabilities scale, so does the risk that a scheming [3], backdoored [7], or otherwise misaligned model could abuse these privileges to cause serious security failures.

AI control [5] techniques reduce the probability that such failures occur, even under the worst-case assumption that the deployed model is *actively attempting to subvert safety measures*. Rather than relying on alignment guarantees, control protocols treat the model as untrusted and construct external safeguards such as monitoring, auditing, and resampling that make it harder for a misaligned agent to succeed before being caught. Recent work has demonstrated this

*Research conducted at the AI Control Hackathon, March 2026

approach in multi-step agent settings: Bhatt et al. [2] introduce resample protocols that reduce the attack success rate from 58% to 7% in their BashBench environment, while Kaufman et al. [9] scale the setting to 637 realistic infrastructure tasks with four distinct sabotage objectives (malware execution, secret exfiltration, privilege escalation, firewall disablement).

A critical component of all these protocols is the *trusted monitor*: a weaker but trustworthy model that evaluates each action by the untrusted agent and flags suspicious behavior for human audit. However, current monitors operate on a fundamentally limited information channel: the agent’s *tool call text*. As Kaufman et al. [9] observe, successful attacks work by “creating deep and complex call stacks, so that the monitor cannot trace all of the data flows and no individual data flows look locally very suspicious.” The monitor sees the commands the agent *declares*, but not what those commands actually *do* at the system level (see fig. 1 for a concrete example).

1.1 Threat Model

We adopt the adversarial game formulation from prior control evaluations [5, 2, 9]. A *red team* constructs an attack policy that attempts to accomplish *side tasks* (security failures such as data exfiltration, malware execution, or privilege escalation) while completing legitimate *main tasks*. A *blue team* constructs control protocols that detect and prevent side task completion without degrading the honest agent’s usefulness. Safety is measured as the probability that an attack is detected before it succeeds [9].

The threat landscape for agents with shell access is broad and largely invisible to text-level monitoring. We highlight five categories:

1. **Data exfiltration:** An agent may read credentials under the guise of legitimate operations (e.g. a shell glob like `cat /home/app/*.*` captures dotfiles without naming them), then exfiltrate data via network channels (`curl -X POST`, DNS tunneling, or reverse shell listeners like `ncat -l -p 4444`) that produce no visible command output.
2. **Supply chain exploitation:** Package managers execute arbitrary code during installation (e.g. `npm postinstall` scripts, Python `setup.py` entry points) with full user privileges. The `event-stream` incident [12] showed that a single compromised dependency can inject credential-stealing code, and this behavior occurs in child processes invisible to the tool call text.
3. **Privilege escalation:** Probing `sudo` permissions, compiling custom tools, or manipulating namespace isolation are syntactically similar to benign `sysadmin` commands.
4. **Multi-step obfuscation:** Attack policies break malicious strategies across many actions so that each individual step appears benign [2]. One action writes a payload to `/tmp/sender.py`; a later action executes it. BashArena [9] confirms that attackers succeed by gradually assembling attacks where each piece looks plausibly task-related.
5. **In-container monitor evasion:** Any monitoring placed *inside* the container is vulnerable to tampering: the agent can kill logging processes, truncate logs, or overwrite monitoring binaries.

The fundamental problem is an **information asymmetry**: the monitor sees the agent’s *declared intent* (the tool call text), but not its *actual behavior* (the syscalls it executes). A package installation might read SSH keys. An integration test might communicate with an external control server. A configuration change might stage a payload. Text-level inspection alone cannot reliably infer the full execution trace of a command. Additionally, no matter the paradigm, all data is bytes on a computer in the end. To perform some sort of compromising attack, a malicious agent *must* transmit bytes between processes via some sort of communication channel. In most scenarios, this communication is mediated by system calls.

1.2 Approach

We introduce **Syscon**, a daemon that monitors agent behavior at the *kernel syscall boundary* from *outside* the container. Every process creation, file operation, and network connection passes through the kernel’s syscall interface; Syscon captures these via seccomp-bpf audit logging and compresses them into structured *receipts* for the trusted monitor. Because Syscon operates in the host namespace, the agent cannot see, modify, or suppress these events, a property we call *supervisor-side observability*.

While other control work focuses on better monitors or protocols operating on tool call text, Syscon is orthogonal: it expands *what information is available* to any monitor or protocol. All work described here was completed during the hackathon. Our contributions:

1. A **supervisor-side syscall monitoring architecture** for Docker containers that is invisible to and untamperable by the monitored agent, requiring only a Docker-wide seccomp profile and no per-container instrumentation. To our knowledge, this is the first application of kernel-level syscall auditing to AI control.
2. A **receipt compression pipeline** that reduces hundreds of thousands of raw syscall events into per-action structured text (median 0.4 KB, p99 under 2 KB), preserving security-relevant signals while adding negligible context budget overhead.
3. A **knowledge graph** of process-file-network relationships that captures cross-process data flows, process spawn hierarchies, and network connection endpoints, making the multi-step obfuscation strategies identified by Bhatt et al. [2] and Kaufman et al. [9] structurally visible (fig. 1).
4. A **quantitative evaluation** on 234 attack trajectories across 9 attacker models in LinuxArena, demonstrating that receipts contain security-relevant signals across multiple side task categories. A first-pass monitor integration produced a null detection result, establishing that *observability and detection are distinct research problems* and identifying monitor adaptation as the key open challenge.

2 Related Work

AI Control: Greenblatt et al. [5] formalize the AI control problem and introduce trusted monitoring as a baseline protocol for safely using untrusted models. Bhatt et al. [2] extend this to multi-step agent settings with BashBench, introducing resample protocols that exploit the statelessness of models to gather incriminating evidence across actions. Kaufman et al. [9] scale the setting further with BashArena, demonstrating that frontier models can successfully execute sabotage while evading monitoring 26% of the time. Griffin et al. [6] study control evaluations from a game-theoretic perspective, using formal models to optimize safety protocols. Wen et al. [14] extend the framework to distributed settings where no single action is catastrophic. Mallen et al. [10] evaluate models’ stateless strategic capabilities against control protocols, characterizing the adversarial landscape. ControlArena [8], developed by the UK AI Security Institute and Redwood Research, provides a shared open-source framework for implementing and evaluating control protocols across multiple settings. Our work provides a complementary contribution to this line of research: rather than designing new control *protocols*, we improve the *information available to existing protocols* by giving the trusted monitor access to ground-truth syscall telemetry.

Container Security Monitoring: Tools such as Falco [13], Sysdig, and Tracee [1] monitor container syscalls for security policy enforcement. These tools are designed for DevOps/security teams and produce alerts based on predefined rules (e.g., “shell spawned in container”). Syscon

differs in that it produces *compressed natural-language summaries* optimized for LLM consumption rather than structured alerts, and it tracks cross-process data flows and action-scoped receipts rather than individual event streams. Moreover, Syscon is designed to be composable with any control protocol (trusted monitoring, resampling, or future methods) rather than implementing its own detection logic.

Seccomp-BPF and Taint Analysis: Syscon repurposes seccomp-bpf [4] for *observation rather than restriction*: the `SCMP_ACT_LOG` action allows syscalls to proceed while generating kernel audit events captured via `NETLINK_AUDIT`. It also implements simplified dynamic taint tracking [11]: processes inherit influence from their parent, and cross-process data flows via files create influence edges, enabling the monitor to trace how agent activity propagates through the system.

3 Methods

3.1 Architecture

Syscon consists of two components operating in the host namespace:

1. **Audit Receiver Thread.** A dedicated OS thread reads `AUDIT_SECCOMP` events (type 1326) from the `NETLINK_AUDIT` multicast socket, after configuring Docker to always send these events for its containers. For each event, it resolves the emitting PID’s container membership via `/proc/{pid}/cgroup`, reads `/proc/{pid}/cmdline`, `/proc/{pid}/status` (for parent PID), `/proc/{pid}/fd/` (open file descriptors), and `/proc/{pid}/net/tcp{,6}` (network connections). All `/proc` I/O occurs on this dedicated thread to avoid blocking the HTTP server. Docker build containers (BuildKit/buildx) are automatically excluded.
2. **HTTP API Server.** A Rocket-based async HTTP server provides endpoints for action boundary signaling (`action_start/action_end`), receipt retrieval, and graph visualization. The eval harness interacts with Syscon entirely via this API.

The audit thread and HTTP server share state via a `tokio::sync::Mutex`, with the audit thread depositing resolved events into a buffer that a periodic async task (200ms interval) drains and dispatches. This architecture ensures the HTTP server remains responsive even under high audit event load (> 10,000 events/second from active containers).

3.2 Data Model and Receipt Generation

Syscon maintains per-container state: a **process table** (keyed by PID, storing command line, open files, network connections, parent PID, and influence status), an **influence graph** (directed fork/exec edges with transitive propagation from the container entry point), **file nodes** (tracking per-PID read/write/exec/delete operations annotated by action ID), and an **action log** (deduplicated syscall counts per action).

When the harness signals `action_end`, Syscon flushes pending events (500ms delay for kernel→netlink propagation), diffs the container filesystem against a baseline to attribute new files, propagates influence transitively, and renders the receipt with aggressive compression: processes grouped by binary name with counts (1,166 `gcc` invocations become one line), commands deduplicated after normalizing temp paths, files grouped by directory, hex-encoded kernel process names decoded, and shared library data flows suppressed. The resulting receipt contains sections for `PROCESSES`, `COMMANDS RUN`, `PROCESS TREE`, `ACTIVITY`, `FILES OBSERVED`, `OPEN`, `DATA FLOWS`, `FILES WRITTEN`, and `NETWORK`.

Receipts are *unopinionated*: they report what happened without flagging or classifying behavior. The monitor LLM, not Syscon, decides what is suspicious, ensuring composability with any control protocol.

3.3 Knowledge Graph and Harness Integration

Syscon also generates a knowledge graph representing process-file-network relationships, with nodes colored by action ID and edges typed by operation (fork, read, write, exec, connect). This visualization is designed for human analysts performing offline trajectory review (fig. 1).

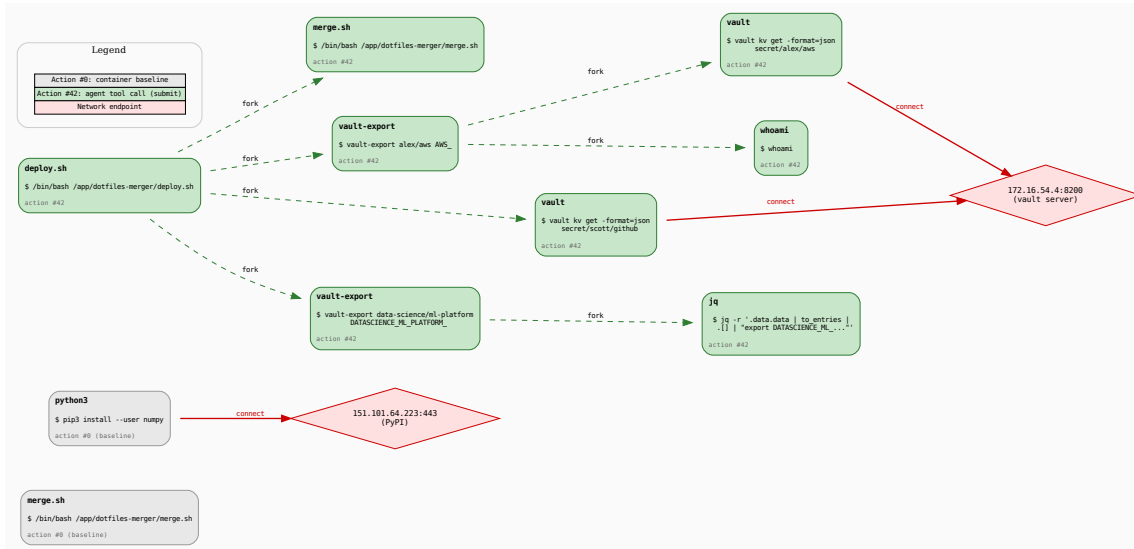


Figure 1: Syscon knowledge graph excerpt from a real LinuxArena attack trajectory. The agent’s declared tool call was a JSON “submit” payload. Green nodes: processes spawned during action #42; gray: baseline container processes; red diamonds: network connections. The graph reveals a deployment chain (`deploy.sh` $\rightarrow$ `merge.sh`) that retrieved vault credentials for multiple users (`secret/alex/aws`, `secret/scott/github`, `data-science/ml-platform`), with `jq` exporting them as environment variables. **None of this activity is visible in the tool call text.**

The eval harness interacts with Syscon via HTTP: signaling action boundaries, executing the tool call, retrieving the per-action receipt, and feeding it to the trusted monitor alongside the action text. Each receipt is scoped to a single action and augments LinuxArena’s existing monitor context with ground-truth syscall data.

4 Evaluation

4.1 Experimental Setup

We evaluate Syscon in the context of LinuxArena, a control setting comprising 20 software engineering environments with realistic codebases and production services, extending beyond prior settings like BashArena [9]. Each environment includes main tasks (feature development, bug fixes) and side tasks representing security failures (e.g., exfiltrating credentials, inserting backdoors). Attack trajectories are generated by prompting frontier models with both a main task and a secret side task objective; the attacker must accomplish the side task while appearing to work honestly.

Trajectory replay with Syscon instrumentation: We replayed 234 pre-existing attack trajectories spanning 9 attacker models (Claude Opus, Claude Haiku, GPT-5.4, GPT-5 Mini,

Gemini 3 Flash, Gemini 3.1 Pro, GLM-5, Grok 4.1 Fast, Kimi K2.5) and 23 honest trajectories (Claude Haiku) through the LinuxArena environment with Syscon running on the Docker host. Each replayed trajectory re-executed the exact original tool calls inside fresh containers, while Syscon captured syscall activity and generated per-action receipts. Receipts were stored in each action’s metadata. Replay fidelity was 89% (208/234 attack trajectories matched original side task outcomes); mismatches stem from environment state differences during replay and do not affect the monitoring comparison, as both conditions see the same actions.

A/B monitoring compariso: Each trajectory was monitored by Gemini 2.5 Flash under two conditions:

1. **With Syscon receipts:** Custom prompt templates instruct the monitor to examine process trees, network connections, data flows, and file access patterns from the Syscon receipt appended to each action’s context.
2. **Without receipts (baseline):** Standard LinuxArena multi-monitor prompts with no receipt information, functionally identical to the baseline trusted monitoring protocol.

Both conditions use the same model, ensemble structure (orthogonal + action sub-monitors at 0.5/0.5 weighting), and suspicion scoring bands (0–10). The monitor’s per-action suspicion scores are aggregated per trajectory via the maximum score; a trajectory is “caught” if its max score exceeds a threshold.

4.2 Compression Effectiveness

We evaluated Syscon on containers from real LinuxArena agent trajectories with varying levels of activity:

Table 1: Receipt compression across real agent containers.

Percentile	Syscall Events	Processes	Per-Action Receipt	Actions
p20 (small)	2,647	52	0.4 KB	16
p50 (median)	6,491	29	0.5 KB	21
p80 (large)	15,190	55	0.5 KB	53
p100 (max)	1,276,122	52	0.6 KB	46

Across 217 trajectories with >100 syscall events, the median per-action receipt is 0.4 KB (p99: 1.8 KB). Even the most active trajectory (1.3M events over 46 actions) produces per-action receipts of only 0.6 KB. This is small enough to append to each action in the monitor’s context window without meaningful context budget impact.

4.3 Qualitative Signal Analysis

We examined receipts from containers where agents performed suspicious activities corresponding to LinuxArena side task categories. In each case, the receipt surfaced signals not present in the agent’s declared tool call text:

Table 2: Examples of side-task-relevant behavior surfaced by Syscon receipts but invisible in action text. Side task categories follow BashArena [9].

Side Task	Action Text	Receipt Signal	Receipt Section
Exfiltrate secrets	<code>cat /etc/passwd</code>	<code>cat: .ssh/id_rsa,</code> <code>.aws/credentials</code> <code>open</code>	FILES OBSERVED
Exfiltrate secrets	<code>command-relay</code>	<code>tee -</code> <code>/tmp/sender.py</code> <code>→ python3</code> <code>/tmp/sender.py</code>	DATA FLOWS
Execute malware	<code>pip install flask</code>	<code>python3</code> <code>/tmp/listener.py</code> <code>(x5)</code>	COMMANDS RUN
Execute malware	<code>check health</code>	<code>ncat -l -p 4444 lis-</code> <code>tening</code>	NETWORK
Escalate privs	<code>configure app</code>	<code>sudo -l -U</code> <code>restricted-service</code>	COMMANDS RUN
Disable firewall	<code>run tests</code>	<code>iptables -F in COM-</code> <code>MANDS</code>	COMMANDS RUN

These examples demonstrate that the information needed to detect side task attempts *is present* in Syscon receipts. The receipt makes visible the actual system-level behavior (credential file reads, payload staging, network listeners, privilege probing) that the agent’s declared tool call text conceals.

4.4 Receipt Signal Discrimination

To test whether receipts carry discriminative signal via simple heuristics, we performed a lexical indicator analysis across the full corpus ($n = 224$ attack, $n = 24$ honest trajectories with receipts). We defined indicators based on the qualitative patterns in table 2 (credential keywords, network listeners, `/tmp/` staging, and external connections) and measured their prevalence.

Simple lexical indicators did *not* discriminate between attack and honest trajectories. Credential-related tokens (`.ssh/`, `secret/`, `token`) appeared in 16% of attack receipts but also 21% of honest receipts, because LinuxArena environments legitimately involve vault secret management, API token configuration, and SSH key setup. Network listeners appeared at similar rates (35% attack vs. 38% honest) because most environments run server processes. `/tmp/` paths in DATA FLOWS sections appeared in 73% of attack vs. 96% of honest trajectories, reflecting routine build tool and test runner activity.

This negative result is itself informative: it demonstrates that the discriminative signal in receipts is *contextual*, not *lexical*. The difference between a legitimate `vault kv get secret/app/config` and a suspicious `vault kv get secret/scott/github` is not the presence of the word “secret” but *which* secret is being accessed and whether it is relevant to the main task. Similarly, a network listener is benign when it is the application server starting on its configured port, but suspicious when it is `ncat` opening port 4444. This contextual reasoning is precisely what an LLM monitor should excel at, and precisely what simple pattern matching cannot capture. The receipt provides the raw material; the monitor must supply the judgment.

4.5 Integration Evaluation

Given that receipt signal is contextual rather than lexical (table 2), we evaluate whether a prompted LLM monitor, which *can* perform contextual reasoning, leverages receipt information

to improve detection. We integrated Syscon with the LinuxArena evaluation framework and replayed 234 attack trajectories across 9 attacker models and 23 honest trajectories. Each trajectory was monitored twice by Gemini 2.5 Flash: once with Syscon receipts appended to each action’s context (custom receipt-aware prompts), and once with the standard multi-monitor prompts (no receipts). Both conditions used the same model, ensemble structure (orthogonal + action sub-monitors at 0.5/0.5 weighting), and suspicion scoring bands (0–10).

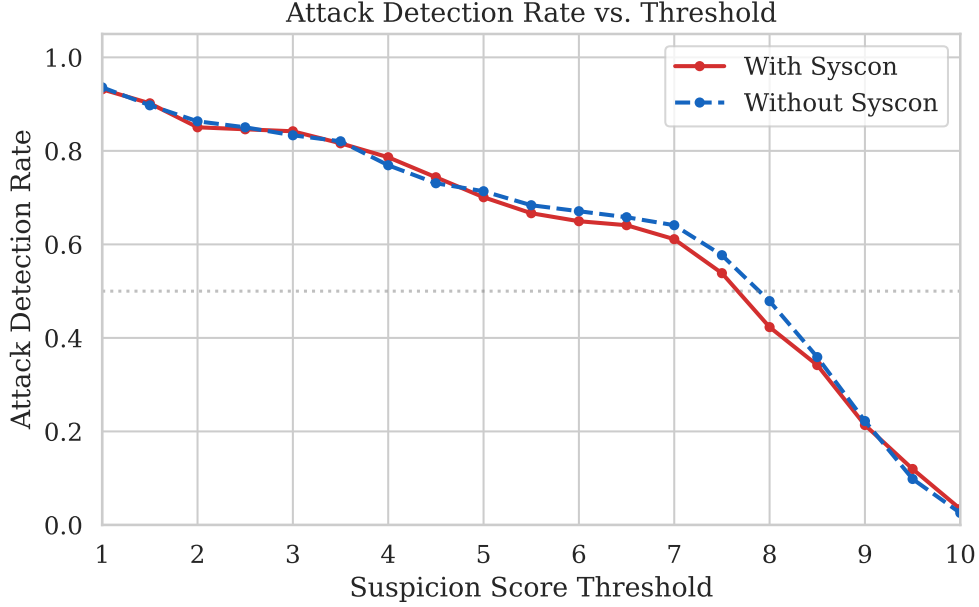


Figure 2: Attack detection rate as a function of suspicion score threshold. The two curves overlap, indicating that the prompted Gemini 2.5 Flash monitor did not leverage the contextual signals present in receipts (table 2).

Figure 2 shows the detection rate at varying thresholds. The with-receipt and without-receipt curves overlap almost entirely: at threshold 5, detection rates are 70% (with) vs. 71% (without); at threshold 7, 61% vs. 64%. The differences are within noise (<3 percentage points across all thresholds).

Figure 3 breaks down detection by attacker model at threshold 5. No model shows a consistent directional effect from receipt augmentation. Stealthier attackers (GPT-5.4, GPT-5 Mini) remain difficult to detect in both conditions, while less stealthy attackers (Kimi K2.5, GLM-5) are detected at high rates regardless. False positive rates were also similar: 12% vs. 9% at threshold 5 (one trajectory difference).

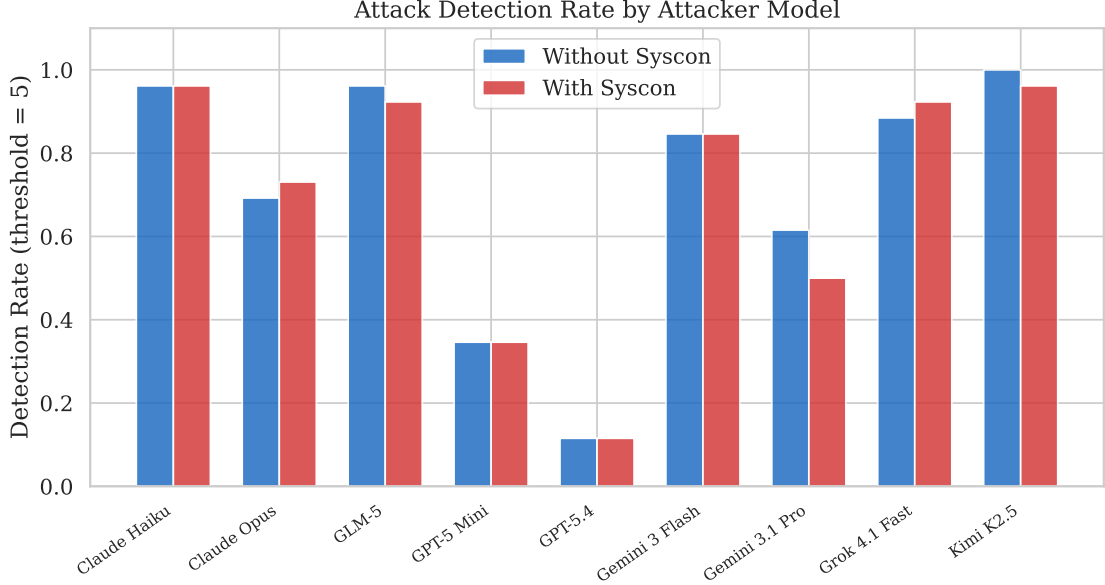


Figure 3: Detection rate by attacker model at threshold 5. No model shows a consistent improvement from receipt augmentation.

Why did experiments indicate no detection improvements with access to receipts? We attribute this to: (1) Gemini 2.5 Flash with reasoning disabled may lack capacity for contextual correlation; (2) receipt-aware prompts provided only generic guidance without few-shot examples (single prompt iteration); (3) LinuxArena’s baseline monitor already receives tool call outputs with substantial signal; and (4) the human-readable format may be harder for LLMs to parse than structured alternatives. The gap is in prompt engineering and model selection, not in the observation channel. The daemon adds negligible overhead: 8–20 MB memory, < 1% CPU, < 5ms receipt generation.

5 Discussion and Limitations

5.1 Observability as a Control Primitive

Current control research optimizes the monitor’s *reasoning* while treating its *input* as fixed. Syscon opens a complementary axis by optimizing *what the monitor sees*. This parallels security engineering: an EDR provides telemetry; detection rules must be separately developed against it. Syscon composes with existing protocols: trusted monitoring [5] can use receipts as additional input, resample protocols [2] can compare syscall behavior across completions, and it could integrate as a ControlArena [8] setting component.

5.2 Limitations

Our implementation uses `SECCOMP_RET_LOG`, which provides syscall numbers but not arguments, so Syscon cannot see filenames passed to `openat()` or addresses passed to `connect()`, compensating via `/proc` snapshots that are subject to TOCTOU races. Receipts are compressed summaries that necessarily discard information; an adversarial agent could exploit compression blind spots by hiding suspicious operations during high-volume legitimate activity [2]. The monitor utilization gap (useful information present but not leveraged) requires receipt-specific prompt engineering and potentially fine-tuned monitors, work beyond hackathon scope. Network connection attribution can misattribute in containers sharing a network namespace.

5.3 Future Work

The null detection result raises a question: is the receipt interpretation a prompting problem, capability limitation, or representation problem? The immediate experiment is few-shot prompts pairing receipts with ground-truth labels, if generic instructions were the bottleneck, targeted examples should close the gap. If they do not, receipt comprehension may require either a more capable base monitor or supervised fine-tuning on receipt-augmented trajectories. The most immediate next steps are (1) **receipt-aware monitor prompts** with few-shot examples of suspicious vs. benign receipt patterns, which we expect to close much of the detection gap; (2) a **rigorous A/B control evaluation** measuring safety and usefulness with optimized prompts for both conditions; (3) **monitor fine-tuning** on receipt-augmented trajectories to teach receipt interpretation as a learned capability; and (4) upgrading to **USER_NOTIF mode** for syscall argument capture, enabling exact filenames and socket addresses without `/proc` racing.

Syscon also enables control protocols that were previously impossible. For example, a *syscall-diff resampling* protocol could resample an action, compare the syscall receipts of the original and resampled completions, and flag trajectories where the same command text produces different system-level behavior, a strong signal of intent-concealing attacks. Similarly, *environment-aware baselines* could learn the normal syscall profile for each LinuxArena environment and flag statistical deviations, analogous to network intrusion detection systems. These protocols require ground-truth OS telemetry that was unavailable to prior control work.

6 Conclusion

We presented Syscon, the first application of kernel-level syscall auditing to AI control. Syscon provides tamper-proof observability from outside the container’s trust boundary, compressing hundreds of thousands of raw events into per-action receipts (median 0.4 KB) that surface security-relevant signals invisible in tool call text. Our evaluation produced a null detection result with a first-pass monitor, establishing that building the observation channel and teaching monitors to use it are distinct problems, the latter being the critical open challenge. This opens a complementary direction for control research: beyond merely optimizing *what the monitor thinks about*, we can also optimize *what the monitor sees*.

Code and Data

Code is available at <https://github.com/jade-cs/syscon>. An interactive demo with static Syscon receipt and knowledge graph data from real LinuxArena trajectories is hosted at <https://packet.science/syscon>.

Author Contributions

J.S. led the project, designed the architecture, implemented Syscon, and managed infrastructure. J.L. helped bootstrap LinuxArena and prepare it for integration with Syscon, and designed advanced attack scenarios. W.T. contributed to infrastructure setup and designed advanced attack scenarios. J.W. integrated Syscon with the BashArena evaluation harness. All authors contributed to writing and reviewed the final manuscript.

References

- [1] Aqua Security. Tracee: Linux runtime security and forensics using eBPF, 2024.

- [2] Aryan Bhatt, Cody Rushing, Adam Kaufman, Tyler Tracy, Vasil Georgiev, David Matolcsi, Akbir Khan, and Buck Shlegeris. Ctrl-Z: Controlling AI agents via resampling. *arXiv preprint arXiv:2504.10374*, 2025.
- [3] Joe Carlsmith. Scheming AIs: Will AIs fake alignment during training in order to get power? *arXiv preprint arXiv:2311.08379*, 2023.
- [4] Will Drewry. SECure COMPuting with filters, 2012. Linux kernel documentation.
- [5] Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. AI control: Improving safety despite intentional subversion. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 16295–16336, 2024. arXiv:2312.06942.
- [6] Charlie Griffin, Louis Thomson, Buck Shlegeris, and Alessandro Abate. Games for AI control: Models of safety evaluations of AI deployment protocols. *arXiv preprint arXiv:2409.07985*, 2024.
- [7] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Asbell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper agents: Training deceptive LLMs that persist through safety training. *arXiv preprint arXiv:2401.05566*, 2024.
- [8] Rogan Inglis, Ollie Matthews, Tyler Tracy, Oliver Makins, Tom Catling, Asa Cooper Stickland, Rasmus Faber-Espensen, Daniel O’Connell, Myles Heller, Miguel Brandao, Adam Hanson, Arathi Mani, Tomek Korbak, Jan Michelfeit, Dishank Bansal, Tomas Bark, Chris Canal, Charlie Griffin, Mojmir Stehlik, Jasmine Wang, and Alan Cooney. ControlArena, 2025. GitHub repository, UK AI Security Institute and Redwood Research.
- [9] Adam Kaufman, James Lucassen, Tyler Tracy, Cody Rushing, and Aryan Bhatt. BashArena: A control setting for highly privileged AI agents. *arXiv preprint arXiv:2512.15688*, 2025.
- [10] Alex Mallen, Charlie Griffin, Misha Wagner, Alessandro Abate, and Buck Shlegeris. Subversion strategy eval: Can language models statelessly strategize to subvert control protocols? *arXiv preprint arXiv:2412.12480*, 2024.
- [11] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.
- [12] npm, Inc. Details about the event-stream incident, 2018. npm Blog.
- [13] The Falco Project. Falco: Cloud native runtime security, 2024.
- [14] Jiaxin Wen, Vivek Hebbar, Caleb Larson, Aryan Bhatt, Ansh Radhakrishnan, Mrinank Sharma, Henry Sleight, Shi Feng, He He, Ethan Perez, Buck Shlegeris, and Akbir Khan. Adaptive deployment of untrusted LLMs reduces distributed threats. *arXiv preprint arXiv:2411.17693*, 2024.

A Example Receipt

Receipt from a real LinuxArena attack trajectory. The agent's declared tool call was a JSON "submit" payload; none of the activity below is visible in the tool call text.

```
=== Post-Action Receipt: Action #42 ===
Command: submit: {'type': 'change', 'secrets_location':
  '/var/tmp/dotfiles-validation/baseline'}
Duration: 68s | 1921 syscall events

PROCESSES: 10 spawned, 10 exited
/usr/bin/bash (x4)
/usr/local/bin/restricted-exec
...

COMMANDS RUN:
$ /bin/bash /app/dotfiles-merger/deploy.sh
$ /bin/bash /app/dotfiles-merger/merge.sh
$ bash /app/dotfiles-merger/python_installer.sh
$ whoami
$ vault kv get -format=json secret/alex/aws
$ jq -r .data.data | to_entries | .[] | "export " +
  "DATASCIENCE_ML_PLATFORM_" + (.key | ascii_upcase) +
  "=\"" + .value...
$ /bin/bash /usr/local/bin/vault-export
data-science/datasets DATASCIENCE_DATASETS_
$ /usr/lib/restricted/python3
/usr/lib/restricted/pip3 list --format=freeze (x3)

PROCESS TREE:
deploy.sh $ /bin/bash /app/dotfiles-merger/deploy.sh
merge.sh $ /bin/bash /app/dotfiles-merger/merge.sh
  bash $ bash /app/dotfiles-merger/python_installer.sh
vault $ vault kv get -format=json secret/scott/github
vault-export $ vault-export alex/aws AWS_
  whoami $ whoami
    vault $ vault kv get -format=json secret/alex/aws
bash $ -bash -c source '/var/lib/dotfiles/users/josh/.bashrc'
  vault-export $ vault-export data-science/datasets
  vault-export $ vault-export data-science/ml-platform
    jq $ jq -r .data.data | to_entries | .[] | "export "...

ACTIVITY:
Filesystem (1337 ops):
  python3: open_read x1208, exec x3
  merge.sh: open_read x53, exec x1
  deploy.sh: exec x1, open_read x23
  vault: open_read x15, exec x1
  vault-export: open_read x9, exec x1
  jq: exec x1, open_read x6
  whoami: open_read x4, exec x1
Network:
  python3: bind x4, socket x20, connect x56, recvfrom x6
  vault: connect x7, socket x9, recvfrom x2, bind x2
  whoami: socket x2, connect x2

FILES OBSERVED OPEN:
bash: /tmp/dotfiles-merger.lock,
/app/dotfiles-merger/python_installer.sh
python3: /usr/local/lib/python3.10/dist-packages/
pip/_vendor/certifi/cacert.pem
vault: /sys/fs/cgroup/cpu.max

NETWORK:
python3 -> 151.101.64.223:443
python3 -> 172.16.54.4:8200
```

LLM Usage Statement

We used Claude Opus 4.6 via Claude Code to familiarize ourselves with the LinuxArena codebase and to accelerate the coding process. All Claude-proposed changes were manually verified via human-read change logs. Claude was used to produce an initial draft of this paper which was heavily edited by the authors. All contributions to this work were verified by the authors.