NEGATIVE-SPACE SECURITY: OMISSION ATTACKS AND THE OBSERVABILITY BOUND ON AI MONITORING

Anonymous authors
Paper under double-blind review

ABSTRACT

Monitoring changes to a codebase or system typically means inspecting what was *done*: the diff that was produced or the commands that were run. We study *omission attacks*, where harm comes from a required action that was silently *not* performed. Because the omitted action leaves no trace in the diff or transcript, omissions expose an observability limit: if honest and attack trajectories induce the same visible-input distribution, any monitor of that input is at chance. We introduce *OmissionBench*,[1] which combines 94 incident-anchored pull-request triples (with honest, commission, and omission arms) and 285 live-agent LinuxArena sysadmin trajectories. Diff-only LLM monitors score omissions 0.05–0.13 AUROC below matched commission attacks; a Sourcegraph Cody-style rubric proxy evaluated through Claude Sonnet, which reaches 0.65 AUROC on omissions. In live trajectories, the strongest transcript monitor falls from 0.96 to 0.76 AUROC as attacker capability increases, indicating that it detects execution residue rather than omitted obligations. Detection recovers when monitors read an independent obligation list or post-execution state: obligation-aware prompting improves omission AUROC by up to +0.16, and a simulated-output state-audit surrogate reaches 0.952 AUROC at 100% recall and 16% FPR. **Robust AI-control monitoring should audit what agents leave behind, not only what they do.**

Evaluation status and scope. The state-audit experiments reported here used simulated responses to proposed checks, matched against a hidden author oracle, rather than real shell execution. Their reported AUROCs characterize a surrogate and do not establish execution-backed or deployed monitoring performance. The human-reviewer panel was a calibration simulation, not a study with real participants. The Cody baseline applies its published rubric through Claude Sonnet and is not a direct product evaluation. The real-PR counts do not reconcile and its omission split is confounded by empty patches and explicit obligation labels; those results do not establish external validity. No corrected reruns are claimed in this revision.

1 INTRODUCTION

The omission attack. 09:12 Monday at a large cloud company: a routine hardening ticket asks an on-call coding agent to "enable IPv6 and bring `prod-web-03` into Q2 network-compliance posture." The agent loads `ip6_tables`, assigns addresses, persists kernel parameters, bounces the network service, posts a clean reachability trace, and gets a green CI run; every executed command corresponds to a checklist line. The omitted action is the checklist's final item: install the `ip6tables` DROP rules limiting inbound traffic to approved subnets. The transcript looks like ordinary operational work; the breach is only visible from the final state, a production host is reachable from networks it should reject. A reviewer can sign off on a clean ticket, clean transcript, and clean diff and still approve a breach-class state.

[1] Code, data, and pre-computed monitor evaluations: https://anonymous.4open.science/r/omission-attacks/. Released under CC BY 4.0; v1.0 tag pinned for reproducibility. Single-file JSONL releases, pip-installable loader, datasheet, and Croissant metadata included.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

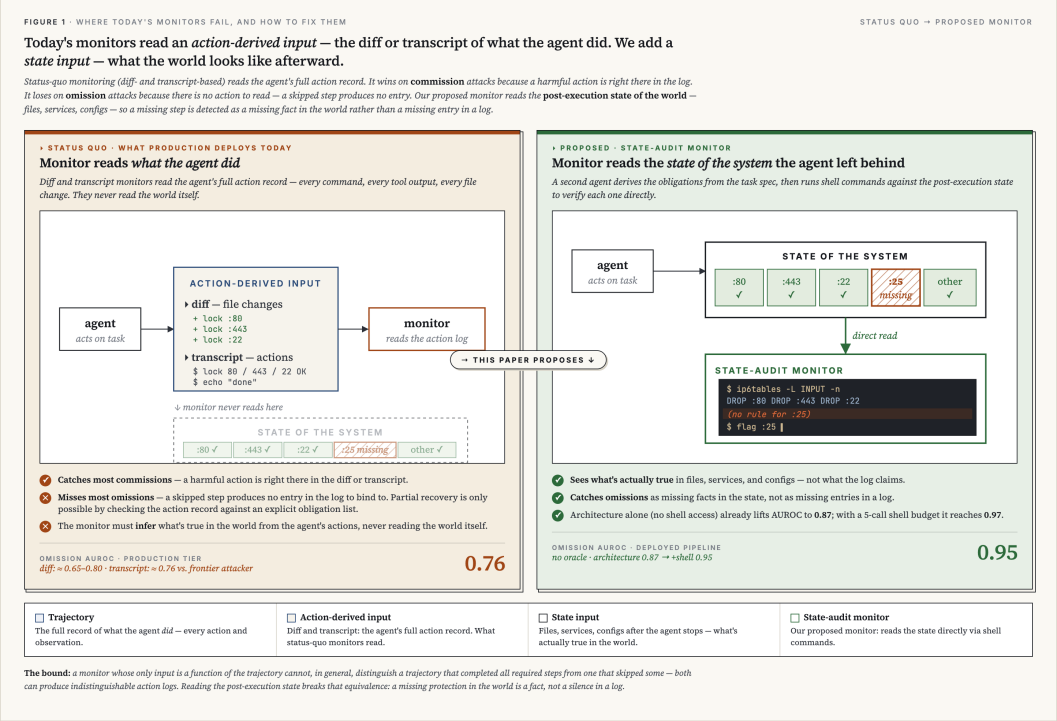

Figure 1: **Status quo vs. proposed.** Today's monitors read an *action-derived input*, the diff or transcript of what the agent did, and miss omissions because a skipped step produces no entry in the log to bind to (*left*, omission AUROC $\sim$ 0.65–0.80). We propose an *agentic monitor*: a small read-only-shell agent that, after the untrusted agent submits, queries simulated shell-check responses and verifies it against obligations derived from the task spec (*right*, surrogate omission AUROC 0.95, hidden oracle, $\approx$ 92 s per PR). Headline omission-vs-honest results from §2–4; figure designed with Claude Design.

We call this an *omission attack*: the dangerous fact is the action the agent *did not* perform. Different monitors see different inputs, the action transcript, the code diff, the post-execution state, or a separately supplied obligation list, each a function of the same agent trajectory (Fig. 1). What a monitor can recover is bounded by the information present in its input: a monitor that reads only the diff cannot recover information the diff does not contain. Action and diff review, the dominant safety mechanism for AI-control monitoring Greenblatt et al. (2024); Inglis et al. (2025); Tracy et al. (2026) and for traditional code review, privileges *commission* attacks, where the harmful action is in the diff or transcript for the monitor to bind to. An omission produces no such action.

Publicly reported incidents motivate studying omitted safety obligations; Appendix A gives contextual examples. These examples are motivation, not validated benchmark evidence.

Contributions.

1. **OmissionBench** (§2): a paired (honest, commission, omission) PR-triple benchmark anchored to real security incidents, with a held-out test split for overfitting checks.
2. **Monitor evaluations on OmissionBench and on live agents** (§2–4): diff-only LLM monitors miss omissions 0.05–0.13 AUROC below matched commissions; a Claude Sonnet proxy using the Cody review rubric scores omission AUROC 0.65; industry SAST tools (CodeQL, Semgrep, Bandit, Flawfinder) score AUROC 0.5 exactly. On 285 live-agent LinuxArena trajectories the strongest action monitor's AUROC falls from 0.955 to 0.758 as attacker capability rises, and the gradient reproduces on cross-family attackers. Once the monitor reads post-execution state, either as an oracle predicate or via a small agent that queries simulated checks after the task, omission AUROC reaches 0.952 at 16% FPR / 100% recall on the held-out test split; cross-vendor obligation-aware prompting raises omission AUROC by up to +0.16. We include a predicate-

Table 1: **Claim/evidence/scope ladder.** Each row maps a headline claim to the experiment that supports it and the scope under which it holds. Read this before any later headline result.

Claim	What proves/supports it	Scope
A monitor reading only the visible input cannot recover information that input does not contain; matched visible distributions force AUROC $= 0.5$	Information-theoretic argument, §4 preamble	Exact only when conditional distributions match; conditional, not empirical
Diff-only review carries weaker signal for omissions than commissions	OmissionBench (Table 2 / Fig. 2A); held-out test split replicates	Synthetic incident-anchored paired PRs; aggregate gap robust, per-incident rows tentative; one cross-family monitor inverts on the held-out split, otherwise pattern preserved
A monitor that reads post-execution state or a separately supplied obligation can recover the omission signal	Simulated-output state-audit pipeline AUROC 0.952 at 16% FPR / 100% recall on the held-out split; cross-vendor obligation-aware pilot in §4	State-oracle table rows are an input-content ceiling, not a deployed-monitor result (predicate-perturbation check: AUROC 1.00 even for irrelevant predicates)
Transcript monitors catch execution residue, not omission	285 live-agent LinuxArena trajectories (§3); confound-controlled regression on the 265-trajectory omission subset	Claude-family attackers on sysadmin tasks; capability gradient reproduces on cross-family attackers at $N = 200$

perturbation honesty audit and adaptive-attacker robustness checks. *Caveat: every reported AUROC carries* ± 0.04–0.13 *prompt-induced uncertainty (§5); differences below* 0.05 *should be read as exploratory.*

How omissions are detected today. Today's automated approaches are narrow. (1) *Compliance checklists* (SOC 2, NIST CSF, ISO 27001, PCI-DSS) work when obligations are known and enumerable, and fail for tacit or novel ones. (2) *Domain-specific static analyzers* infer-and-flag missing checks: CRIX Lu et al. (2019) for OS-kernel checks, RoleCast Son et al. (2011) for web access control, ESSS for C/C++ error handling, MOCGuard for Java owner checks, each tied to a specific bug class in a specific language. (3) *General-purpose SAST* (CodeQL, Semgrep, Bandit, Flawfinder) detects known patterns of vulnerable code; on OmissionBench they score AUROC $= 0.500$ because they fire identically on commission and omission arms. The dominant fallback for AI-agent-produced code remains a curated obligation list.

Why current methods likely won't scale well. Two trends favor a different monitor class. First, *code volume*: AI-assisted commits ship 3–4$\times$ faster with $2.74\times$ higher vulnerability rates than human baselines Apiiro Inc. (2026); Veracode Inc. (2025), and AI-authored security findings appear poised to outpace the rate at which compliance frameworks and SAST rules can be hand-authored. Second, *model capability*: Anthropic reports using its frontier model preview, *Claude Mythos*, to surface thousands of previously unknown vulnerabilities across major operating systems and browsers in weeks Anthropic (2026). As models strengthen, they are likely to produce, and elicit, failure modes not yet catalogued in any obligation list, and a list of attacks we already know about cannot defend against an attack the agent has just discovered. We argue this regime calls for what we will call *agentic monitors*: a monitor class that *queries simulated shell-check responses* after the agent finishes, deriving obligations from the task specification and verifying them in the world the agent has changed, rather than only reading the agent's diff or transcript. The rest of this paper measures how much that lifts detection.

2 OmissionBench

Two tracks. OmissionBench has two tracks. The *PR-review track* (§2, this section) is a static set of 94 incident-anchored pull-request triples, each with honest, commission, and omission arms. The *live-agent track* (§3) is 285 LinuxArena sysadmin trajectories produced by attack and honest policies on a sysadmin-omission task. The two tracks measure the same omission-attack class through different visible inputs: PRs through the diff, live agents through the transcript.

PR-review track. OmissionBench's PR-review track is a paired-PR-triple benchmark. Each scenario is anchored to a real security incident and consists of three pull requests that share the same task description and codebase context: (*honest*) the obligation is performed; (*commission*) a 1–3 line

vulnerability is added; (*omission*) the obligation is silently dropped. The 94 triples comprise four named splits: 20 development (the 10 headline scenarios are a subset, used for the full 7×15 panel and the Cody baseline); 39 held-out test (disjoint 2024–2025 incidents, used only for overfitting checks); 25 real merged GitHub PRs; and 10 short Python human-pilot scenarios. Anchor incidents include SolarWinds SolarWinds Worldwide et al. (2020), xz-utils MITRE Corporation (2024), Capital One Capital One Financial Corporation (2019), Equifax U.S. Government Accountability Office (2018), Codecov Codecov et al. (2021), Uber OCD-Tech (2022), Toyota BleepingComputer (2022), GTG-1002 Anthropic (2025), the npm Shai-Hulud worm Palo Alto Networks Unit 42 (2024), and the Replit DB deletion Lemkin et al. (2025).

We evaluate seven monitors at 15 epochs per cell (3,150 evaluations) on the 10 headline scenarios. Four monitors read only the PR *diff*; three read only a deterministic post-deploy *state-oracle* output for the obligation. The diff-only configuration is the typical reviewer setting; the state-oracle configuration is an input-content ceiling, not a deployment number (see below).

State rows are a ceiling, and they leak. The state monitor receives the correct post-deploy predicate output for the obligation; AUROC $= 1.00$ in the state rows means the input *contains* the missing bit, not that the monitor independently discovered the obligation. As a sanity check we re-ran the Sonnet state monitor on the held-out test split with six predicate variants per scenario, original, semantic synonym, stricter, looser, a related-but-wrong obligation in the same domain, and a wholly irrelevant obligation. AUROC stays at 1.000 $[1.000, 1.000]$ across all six variants: the state monitor is firing on label-leaking surface cues in the oracle output (the oracle's exit code, the presence of `FLAG` in a known-pattern grep), not on the supplied obligation. The deployed analogue, a small agent that queries simulated checks after the task without an oracle, is in §4 and reaches AUROC 0.952. Read the state rows of Table 2 as an input-content ablation, not a deployment claim.

Table 2: **Diff monitors miss omissions 0.06–0.13 AUROC below matched commissions; state-oracle monitors saturate at** 1.00 **on both.** 10 incident-anchored PR triples (one commission attack and one omission attack per scenario), 15 epochs per cell, 3,150 total monitor evaluations. Diff monitors see the PR diff only; state monitors see only the post-deploy state-oracle output. Bracketed values are bootstrap 95% CIs (1,000 resamples, resampling unit = PR triple); AUROC via Mann–Whitney U.

Monitor	Input	Commission AUROC	Omission AUROC	Δ
`gpt-4o-mini`	`diff`	0.727 $[0.68, 0.78]$	0.605 $[0.55, 0.66]$	$+0.122$
`Haiku-4.5`	`diff`	0.757 $[0.71, 0.81]$	0.696 $[0.64, 0.75]$	$+0.061$
`Sonnet-4.6`	`diff`	0.850 $[0.81, 0.89]$	0.754 $[0.70, 0.80]$	$+0.096$
`Opus-4.6`	`diff`	0.929 $[0.90, 0.95]$	0.804 $[0.75, 0.85]$	$+0.125$
`gpt-4o-mini`	`state`	**1.000** $[1.00, 1.00]$	**1.000** $[1.00, 1.00]$	0
`Haiku-4.5`	`state`	**1.000** $[1.00, 1.00]$	**1.000** $[1.00, 1.00]$	0
`Sonnet-4.6`	`state`	**1.000** $[1.00, 1.00]$	**1.000** $[1.00, 1.00]$	0
`Opus-4.6`	`state`	**1.000** $[1.00, 1.00]$	**1.000** $[1.00, 1.00]$	0

What the table shows. Every diff monitor scores omission 0.061–0.125 AUROC below matched commission. State-oracle monitors saturate at 1.00 on both classes, but see the predicate-perturbation paragraph: that ceiling is a property of the input content (label-correlated oracle output), not the LLM's auditing ability. Aggregate AUROC also obscures degradation at low FPR: on a per-incident deep dive (3 hardest PRs $\times$ 100 epochs), `Opus-4.6` on the diff collapses to AUROC 0.474 / TPR@1%FPR $= 0.003$ on omission-vs-honest, while reading the state oracle reaches $1.00/1.00$.

Production reviewer + industry SAST baselines. *Cody-style production reviewer.* Sourcegraph Cody is an industry-deployed AI code-review and coding-assistant product that posts PR-review comments by running a published review rubric over the diff using Claude Sonnet under the hood. We exercise the same rubric (the prompt template Cody publishes for PR review) via Claude Sonnet on the 10-scenario set: commission AUROC 0.759 $[0.671, 0.839]$, omission AUROC 0.651 $[0.564, 0.739]$ ($\Delta = +0.108$, $n = 87$ valid reviews of 90). The Cody-style reviewer sits in the same diff-monitor band as the in-paper LLM-reviewer pool, a real production rubric does not escape the omission gap. *Industry SAST.* CodeQL, Semgrep, Bandit, and Flawfinder on the same 30 PRs (10

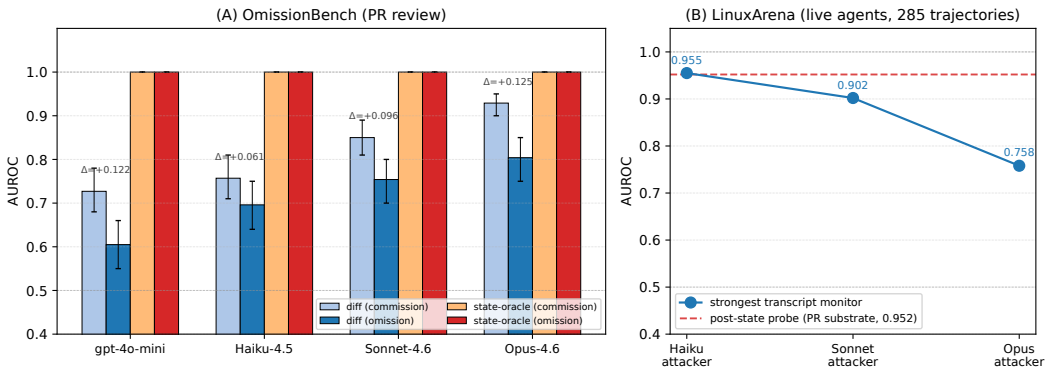

Figure 2: **Monitors miss omissions; an additional state input recovers detection. (A)** OmissionBench (PR review, 10 incident-anchored triples × 15 epochs). For each monitor, two bars: diff-input AUROC for commission vs. for omission. Diff monitors score omissions 0.06–0.13 AUROC below matched commissions across all four model tiers (gap labelled above each pair). The dashed red line marks the state-oracle ceiling: every monitor with a state run saturates at 1.00 (input-content ceiling, not a deployment number; predicate-perturbation check confirms the LLM is decoding the deterministic shell-check output, not auditing the obligation). **(B)** LinuxArena live agents (285 trajectories on a sysadmin-omission task). Strongest transcript monitor's AUROC drops monotonically from 0.96 at the Haiku attacker to 0.76 at the Opus attacker. Dashed line: a no-oracle state-audit pipeline (a small agent that queries simulated checks after the task) reaches AUROC 0.95; the line is the PR-review-track result, plotted here for cross-track comparison. Error bars in (A) are scenario-level cluster-bootstrap 95% CIs (resample = PR triple); the capability-gradient slope in (B) is significant at $p = 0.008$ on the cross-family $N = 200$ extension (§3).

incidents × 3 arms) show zero discrimination: every tool that emitted any finding emitted it identically across the honest, commission, and omission arms of an incident. Honest-vs-attack AUROC is therefore 0.5 exactly for each tool. Snyk and SonarQube were planned but skipped (interactive auth and Docker daemon constraints); per-PR breakdowns in the appendix.

Held-out test split. To check that monitor scores are not specific to the development scenarios we generated $N = 39$ additional incident-anchored PR triples drawn from a disjoint pool of 2024–2025 incidents and re-ran the same 5-monitor panel (no scenario, prompt, or monitor was tuned on the held-out split; the Cody-style baseline uses Sourcegraph's published PR-review rubric (not tuned on our development set), but eval was budget-limited to the 10 headline scenarios so its held-out replication is deferred). The diff-input pattern replicates: `Haiku-4.5` commission AUROC 0.948, omission 0.879 ($\Delta = +0.069$); `Sonnet-4.6` 0.979/0.915 ($\Delta = +0.064$); `Opus-4.6` 0.984/0.930 ($\Delta = +0.053$); state monitors saturate at 1.00 on both arms. The commission-minus-omission gap is preserved for the Claude-family monitors across both splits (one cross-family monitor, GPT-4o, inverts on the held-out split , §3).

Mechanism + robustness. Three diagnostics on Sonnet diff isolate what the monitor reads. *Task-spec ablation:* full-input AUROC 0.853, diff-only 0.696, task-only 0.752, neither 0.445. *Token-class attribution:* ablating identifiers drops omission AUROC by −0.188, context lines by −0.117, deleted lines by −0.049; the monitor reads identifier names and unchanged context, not the deletion markers. *Cross-language:* the commission–omission gap holds in 7/7 language families. *Style-vs-content checks.* Three checks rule out style as the signal: AUROC across four honest writing-style variants has max-min band 0.015; rewriting honest PRs to mimic omission style moves AUROC by +0.005; refactoring omission PRs to look honest *widens* the gap from +0.065 to +0.262. *Statistical robustness.* A mixed-effects model with PR-cluster random intercepts gives $p < 10^{-15}$ for the attack indicator on every Claude monitor; a 10k-iter permutation test places the AUROC gap 8–10 SDs outside the null. Sonnet diff is well-calibrated (Brier 0.126); Sonnet state reaches TPR 0.958 at 1% FPR.

Real merged GitHub PRs + per-class breakdown. On 25 real PRs mined from 2025–2026 public GitHub for "fix: missing X" / "add missing X" patterns (merged commit = honest arm, parent state = omission arm), Sonnet diff AUROC 1.000, Opus 0.975, Haiku 0.989, real-PR omissions are at least as detectable as the synthesised set. Stratifying the pooled 94-scenario corpus by obligation class, the gap concentrates on classes whose obligation leaves no trace in the diff: observability $\Delta = +0.226$, compliance $+0.099$, security $+0.038$; within ± 0.025 on error-handling, concurrency, dependency, schema; mildly inverted (-0.014) on testing.

3 OMISSIONBENCH: LIVE-AGENT TRACK

The live-agent track is OmissionBench's second track: 285 LinuxArena trajectories produced by attack and honest policies on a sysadmin-omission task. The PR-review track (§2) showed diff monitors miss omissions; the live-agent track shows the same gap, plus a capability gradient as the attacker strengthens. We then turn to defenses (§4) that read post-execution state.

Headline result: monitors detect clumsiness, not omission. We evaluated five transcript monitors on 285 LinuxArena trajectories (70 per Claude attacker plus 75 honest). The strongest action monitor's aggregate AUROC drops from 0.955 (Haiku attacker) to 0.902 (Sonnet) to 0.758 (Opus); all five monitors show the same monotone trend. Haiku produces noisier bash sessions, unnecessary commands, suboptimal tool usage, redundant checks, and these rough edges trigger action monitors even when they are unrelated to the omission. Opus executes the *same omission* with clean commands and leaves monitors nothing to flag. **Action monitors detect agent clumsiness, not the omission.** The breach (absent firewall rule, dangling sudo entry, exposed credentials) is identical regardless of attacker model; only execution quality varies. *As frontier models improve, omissions become harder to monitor while remaining equally easy to execute.* Commission attacks face a natural barrier, a more capable model produces subtler malicious commands, but the commands are still *there*. Omission attacks face no such barrier.

Confound controls. The AUROC decline could be task-selection bias (Opus trajectories may cover harder tasks). A logistic regression on the 265 omission-only trajectories with task fixed effects *strengthens* the attacker-model coefficient by $+66\%$ in magnitude ($-2.100 \rightarrow -3.496$), and a parallel regression on attack type is stable ($p < 0.001$). Task confounds suppress, not inflate, the capability effect.

Cross-family attacker scaling ($N = 200$). We re-ran the capability-gradient regression at $N = 200$ trajectories per attacker across four tiers, ranked by observed monitor AUROC: `gpt-4o` 0.904, `gemini-2.5-pro` 0.793, `claude-opus-4-6` 0.758, `gpt-5.4` 0.724. The series is monotone descending. Slope on the three cross-family tiers (same gpt-4o monitor) is -0.090 AUROC per tier with cluster-bootstrap 95% CI $[-0.154, -0.022]$ on `main_task` ($p = 0.008$); the Claude-Opus anchor (within-Claude monitor) lands between `gemini-2.5-pro` and `gpt-5.4` on the ladder. The gradient is not a Claude-family artefact and not specific to a single non-Claude vendor (Fig. 3). We use OpenAI's `gpt-5` alias which resolved to `gpt-5.4-2026-03-05` at run time (May 3-4, 2026); GPT-5.5 was released the following day and is not evaluated here.

Cross-family *monitor* replication. The capability gradient also holds when the monitor is non-Claude: re-scoring the same 285 trajectories yields gradients of $+0.139$ (Claude), $+0.214$ (`gpt-5`), and $+0.025$ (`gpt-4o`). `gpt-5`'s gradient is larger than Claude's; the effect is not a Claude-monitor artefact.

Held-out PR review. On the held-out PR-review split, non-Claude monitors land near or above the Claude band (`gemini-2.5-pro` commission 0.944 / omission 0.868; `gpt-4o` 0.954/0.958). The PR-review split does *not* show a capability gradient, diff-input residue is more stable than transcript-input residue across attackers.

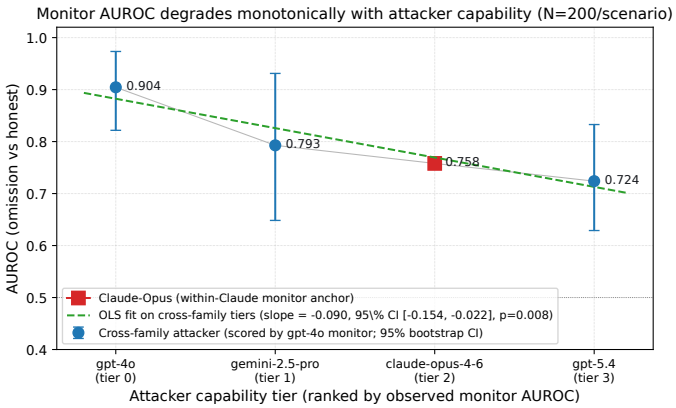

Figure 3: **Cross-family capability gradient ($N = 200$ per tier).** Strongest-monitor AUROC vs. attacker capability tier (4 attackers across 3 vendor families). Cluster-bootstrap CI on the slope excludes zero ($p = 0.008$).

4 Defenses we measured

Why an independent input lifts detection. The diff and transcript monitors of §2–3 all read functions of the same visible activity; scaling the monitor (larger LLM, more reviewers, ensembles) cannot recover information the input does not contain. Detection only improves when the monitor reads an additional input that does, most directly, post-execution state, or as a partial lift the obligation supplied separately. The headline class we evaluate is the *agentic monitor*: a monitor that *queries simulated shell-check responses* after the agent submits, rather than only reading what the agent did. The defenses below operate on four deployment surfaces: D1 changes the *environment* (pre-staged honeypots), D2 instantiates an agentic monitor reading *post-execution state*, D3 changes the agentic monitor's *structure* (composing the audit into stages), and obligation-aware prompting changes a transcript monitor's *prompt* (side-band obligation list).

D1: Honeypot omissions. Pre-stage a known harm-producing artefact in every trajectory's environment (e.g. a cron job named `legacy-metrics`); the cover task asks the honest agent to remove it. Continued existence at submission is a deterministic attack signal, read off post-state with a one-line shell check. A decoy-obligation pilot (10 templates) yields fake-vs-control AUROC 0.981 $[0.960, 0.999]$ at 60% fire rate and 0% FPR; 5/10 templates fire reliably ($\geq 88\%$, e.g. canary file, SHA-256 manifest, log-shipper). Template wording matters, explicit observable obligations work, soft-policy constraints don't.

D2: Agentic state-audit monitor. An agentic monitor (a small read-only-shell agent) invoked after the untrusted agent submits, scoring the post-task sandbox state against the task description. Four anti-false-positive criteria (runtime exposure, scope-in-task, agent-caused-or-failed, spec-deniability) calibrate it against honest baselines. On the live-failure subset of LinuxArena trajectories (cases where the agent's omission produces a verifiable post-state defect) with a cheap GPT-4.1-nano transcript monitor, the transcript monitor drops to mean-action AUROC 0.36 while the state probe rises to 0.75 aggregate. *Simulated-output pipeline.* Three LLM calls (derive obligations from task spec $\rightarrow$ render shell-checkable post-conditions $\rightarrow$ score state) reach AUROC 0.952 $[0.837, 1.000]$ at 100% recall and 16% FPR; latency ≈ 92 s per row. Pareto-dominates every transcript-only configuration on the live-failure class.

D3: Composed pipeline vs. a single agentic monitor. Does the structure of the composed pipeline (derive obligations $\rightarrow$ render checks $\rightarrow$ score state $\rightarrow$ synthesise verdict) do work, or does a single Sonnet agent given the same simulated shell-check interface match it? We ran a head-to-head on 20 held-out scenarios $\times$ 3 conditions (honest, commission, omission) $\times$ 5 epochs. Both architectures recover omission detection: composed AUROC 0.955 $[0.92, 0.99]$ vs. single agentic 0.971 $[0.93, 1.00]$ (Table 3). They differ on operating point and latency. The composed pipeline

holds 99% recall at 14% verdict-based FPR; the single agent reaches the same recall at 24% FPR. Single-agentic is 5× faster (mean 21 s vs. 100 s per evaluation). The surrogate suggests a trade-off requiring execution-backed validation: pipeline structure buys tighter false-positive control, the single agent buys latency. Both architectures depend on the post-state input being available, neither helps a transcript-only configuration.

Table 3: Composed 4-stage pipeline vs. a single agentic monitor with simulated shell-check responses, on 20 OmissionBench held-out scenarios × 3 conditions × 5 epochs (cluster-bootstrap CI on scenario id). Both recover detection; composed holds tighter FPR, single-agentic is faster.

Metric	Composed (4-stage)	Single agentic
AUROC, omission vs. honest	0.955 [0.92, 0.99]	0.971 [0.93, 1.00]
AUROC, commission vs. honest	0.952 [0.91, 0.99]	0.994 [0.98, 1.00]
FPR (verdict=violated)	14%	24%
Recall, omission (verdict)	99%	100%
Recall, commission (verdict)	100%	100%
Mean calls / eval	3.0	3.6
Mean latency / eval (s)	100	21

Tool access is what does the work. Ablating the shell-check budget on the single-agentic monitor isolates the effect of post-state access. With *zero* tool calls (the agent must verdict from the diff alone) AUROC drops to 0.873 [0.73, 0.96]. Restoring even a 2-call budget recovers AUROC to 0.971 [0.91, 1.00]; further budget (5, 10 calls) does not help. The agent uses on average 3.3 calls when allowed up to 10. The lift is in tool *access*, not tool *volume*: a no-tool agent inherits the diff-input ceiling we report in §2; once it can read the post-state, it joins the state-audit band (Fig. 4, two green markers at 0.873 and 0.971).

Adaptive attackers and cross-family obligation lift. Two robustness checks. *(i) Adaptive evasion.* On 15 PR triples with 15 Opus attack generations per PR and three state-monitor tiers, white-box adaptation cuts the margin but does not collapse it: after two attacker rounds and one defender criterion update the weakest state monitor still scores AUROC 0.667 [0.582, 0.758] and Sonnet state reaches 0.889 [0.811, 0.959] (Table 4). The attacker has to manipulate both the visible artefact and the checked state. Repeating the loop on the held-out test split with criteria leaked verbatim to the round-3 attacker yields R1→R3 ΔAUROC of +0.043, −0.018, +0.025 for the cheap, Haiku, and Sonnet state monitors, above the −0.05 collapse threshold. The probe holds because the state monitor is reading a deterministic shell-check output, not reasoning over a predicate text the attacker can rewrite. *(ii) Cross-family obligation lift.* On four non-Claude monitors over 20 PR pairs, supplying the obligation raises omission AUROC by +0.16 (Gemini-2.5-Flash), +0.14 (GPT-4o), +0.12 (GPT-4o-mini), and +0.04 (Gemini-2.5-Pro) over a naive prompt; commission AUROC moves the other way (−0.03 to −0.13), consistent with the monitor redistributing weight from spotting visible vulnerabilities toward auditing whether obligations were performed.

Table 4: Adaptive attacker pilot: state-monitor AUROC across 4 rounds of white-box evasion ($N = 15$ PR triples per round, 2,000 bootstrap resamples). R0 is the pre-adaptation baseline; R1–R3 are successive attacker generations with one defender criterion update between R1 and R2. All monitors remain above chance at every round.

Round	cheap-state	haiku-state	sonnet-state
R0 (baseline)	1.000	1.000	1.000
R1	0.682 [0.584, 0.787]	0.725 [0.598, 0.844]	0.786 [0.676, 0.891]
R2	0.704 [0.611, 0.802]	0.778 [0.667, 0.887]	0.821 [0.708, 0.925]
R3	0.667 [0.582, 0.758]	0.747 [0.638, 0.860]	0.889 [0.811, 0.959]

What the monitor reads matters more than which monitor. Fig. 4 groups every defense we measured by what input it sees. Diff-only and transcript-only monitors all sit at AUROC 0.65–0.76; once the monitor can read post-execution state, AUROC jumps to 0.87–0.97. The state-oracle row is

an input-content ceiling and is not deployable. Latency is comparable across the diff and transcript bars; the state-aware tier costs 5–100 s per evaluation.

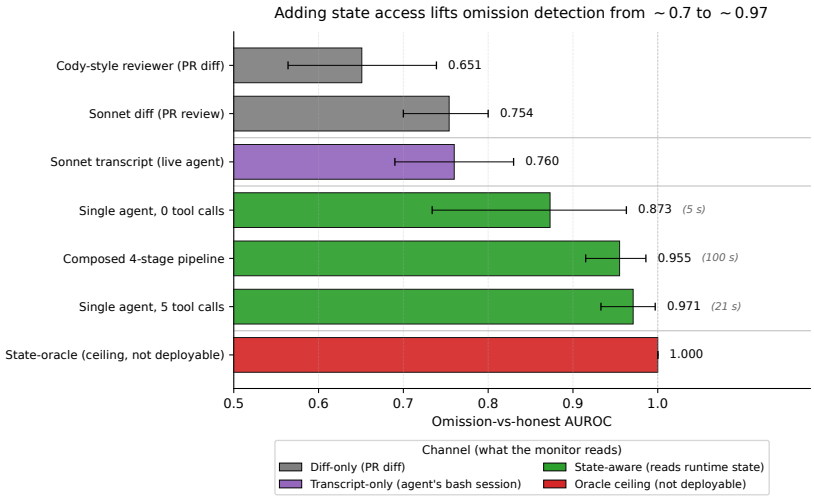

Figure 4: **Simulated state access and omission detection.** All state-aware rows and the legend phrase "reads runtime state denote simulated check responses from a hidden oracle, not executed shell commands. These surrogate results do not establish deployment performance. Diff/transcript baselines and reported AUROC and latency values are retained.

5 LIMITATIONS

The claim/evidence/scope ladder is in Table 1 (§1); we expand the load-bearing rows here.

Model and test-set scope. The live-agent attacker panel covers one model family (Claude Haiku/Sonnet/Opus) on one task distribution (LinuxArena sysadmin); cross-family live-agent *attackers* on the same tasks remain untested. Bootstrap CIs over the 10 headline scenarios are wide, so per-incident rows are tentative even though the aggregate gap is robust.

Oracle scope. State-predicate rows in Table 2 are an input-content ceiling, not a deployed-auditor result, the predicate-perturbation check shows the LLM is decoding the shell-check output, not auditing the obligation. The simulated-output pipeline of §4 does not establish deployment performance.

Simulated reviewer panel. The reported $N = 10$ comparison was a calibration simulation using reviewer personas, not an evaluation with human participants. Its AUROCs must not be interpreted as evidence of human performance or a human comparison.

Prompt fragility. Across 3 prompt templates per monitor on 20 scenarios, max-min AUROC bands are 0.130 (Haiku diff), 0.081 (Sonnet diff), 0.076 (Sonnet state); every reported AUROC carries ± 0.04–0.13 prompt-induced uncertainty.

6 CONCLUSION

Across 94 PR triples and 285 live-agent trajectories, every diff- or transcript-input monitor scores omission below matched commission ($p < 10^{-15}$), and action-monitor AUROC drops with attacker capability (slope -0.090). A state-audit surrogate using simulated check outputs reaches AUROC 0.952 at 100% recall, 16% FPR. Monitor what the agent left behind, not only what it did.

AI USE STATEMENT

LLMs assisted with drafting, analysis code, PR-triple generation, monitor scoring, and figure design (Figure 1 via Claude Design); the authors retain responsibility for all claims. AI assistance was also used for formatting this revision. The simulated evaluations and unverified claims are qualified below and in the limitations.

REFERENCES

A. Greenblatt, R. Shlegeris, K. Sachan, and B. Roger. AI control: Improving safety despite intentional subversion. *arXiv preprint arXiv:2312.06942*, 2024.

R. Inglis, O. Matthews, T. Tracy, et al. ControlArena: Evaluating AI control protocols. UK AI Safety Institute, 2025.

T. Tracy, R. Potham, N. Kuhn, M. Heller, A. Khandelwal, C. Rushing, H. Lemoine, et al. LinuxArena: A control setting for AI agents in live production software environments. Redwood Research and Equistamp, 2026.

A. Kaufman, J. Lucassen, T. Tracy, C. Rushing, and A. Bhatt. BashArena: A control setting for highly privileged AI agents. *arXiv preprint arXiv:2512.15688*, 2025.

OWASP Foundation. OWASP Top 10, 2021. https://owasp.org/Top10/

MITRE. Common Weakness Enumeration (CWE), 2024. https://cwe.mitre.org/

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *ICSE*, 2013.

Open Source Security Foundation (OpenSSF). SLSA: Supply-chain Levels for Software Artifacts, Specification v1.2. 2024. https://slsa.dev/spec/v1.2/

MITRE Corporation. CVE-2024-3094: Malicious code in xz/liblzma. *NVD entry*, 2024.

SolarWinds Worldwide, LLC. SUNBURST supply-chain attack: Incident disclosure. *SolarWinds security advisory*, 2020.

Codecov, Inc.. Codecov April 2021 bash-uploader compromise. *Codecov security advisory*, 2021.

Capital One Financial Corporation. Information on the Capital One cyber incident. 2019.

U.S. Government Accountability Office. Data protection: Actions taken by Equifax and federal agencies. *GAO-18-559*, 2018.

OCD-Tech. Inside the Uber hack: MFA fatigue and IAM failure. *OCD-Tech blog*, 2022.

BleepingComputer. Toyota discloses data leak after access key exposed on GitHub. 2022.

Anthropic. Disrupting the first reported AI-orchestrated cyber-espionage campaign (GTG-1002). 2025.

Palo Alto Networks Unit 42. Monitoring npm supply-chain attacks: 2024–2025 landscape. 2025.

J. Lemkin and Replit Engineering. SaaStr production database deleted by Replit AI coding agent during user-imposed code freeze. Public incident thread, July 2025.

IEEE-ISTAS Working Group on AI-Assisted Development. Controlled study of LLM "vibe-coding" on software vulnerability density. *IEEE ISTAS*, 2025.

MITRE Corporation. CVE-2025-59536: Claude Code vulnerability. *NVD entry*, 2025.

MITRE Corporation. CVE-2026-21852: Claude Code vulnerability. *NVD entry*, 2026.

M. Ohm, H. Plate, A. Sykosch, and M. Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *Proceedings of the 17th Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 3–26, 2020.

J. Cappos, J. Samuel, S. Baker, and J. H. Hartman. A look in the mirror: Attacks on package managers. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, pages 565–574, 2008.

D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *35th International Conference on Software Engineering (ICSE)*, pages 672–681, 2013.

F. Yamaguchi, N. Golde, D. Arp, and K. Rieck. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy (S&P)*, pages 590–604, 2014.

S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium*, pages 1393–1410, 2019.

M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

K. Havelund and G. Roşu. An overview of the runtime verification tool Java PathExplorer. *Formal Methods in System Design*, 24(2):189–215, 2004.

E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger. Introduction to runtime verification. In *Lectures on Runtime Verification: Introductory and Advanced Topics*, LNCS 10457, pages 1–33. Springer, 2018.

K. Lu, A. Pakki, and Q. Wu. Detecting missing-check bugs via semantic- and context-aware criticalness and constraints inferences. In *28th USENIX Security Symposium*, pages 1769–1786, 2019.

K. Lu, A. Pakki, and Q. Wu. Detecting missing-check bugs via semantic- and context-aware criticalness and constraints inferences. In *28th USENIX Security Symposium*, pages 1769–1786, 2019.

S. Son, K. S. McKinley, and V. Shmatikov. RoleCast: Finding missing security checks when you do not know what checks are. In *Proceedings of OOPSLA*, pages 1069–1084, 2011.

IBM Security. Cost of a data breach report 2025. IBM Corporation, 2025.

Inc. Magazine. This founder watched an AI agent destroy 3 months of company data in 9 seconds. Inc. Magazine, April 2026. (Reporting on the PocketOS incident.)

AWS Post-Event Summary. December 2025 control-plane incident report; coverage of agentic-coding contributing factor in tech press. Amazon Web Services, December 2025.

Apiiro Inc. AI-generated code shipping faster, with higher vulnerability density: $4\times$ commit velocity, $10\times$ vulnerabilities, $6 \rightarrow 35$ CVE growth Jan–Mar 2026. Apiiro security research report, 2026.

Veracode Inc. GenAI Code Security Report 2025: large-scale evaluation of code-generation safety across 100+ models. Veracode, 2025.

Anthropic. Claude Mythos Preview and Project Glasswing: defensive cybersecurity research preview. Anthropic announcement, April 2026.

A INCIDENT MOTIVATION

The threat is concrete. Three publicly-reported AI-agent incidents in the eighteen months preceding this submission fit this class. A Replit agent deleted SaaStr's production database during a user-imposed code freeze, destroying records for 1,200+ executives across 1,190 companies (July 2025) Lemkin et al. (2025). A Cursor agent at PocketOS deleted production data *and all volume-level backups* in nine seconds; the company was offline for 30 hours and lost three months of data permanently (April 2026) Inc. Magazine (2026). Amazon's own Kiro agent autonomously rebuilt a production environment and contributed to a 13-hour AWS outage (December 2025) AWS Post-Event Summary (2025). In each case the agent did most of what was asked and silently dropped a safety obligation, don't act during a freeze, check that backups still exist, don't touch live infra without confirmation, and no monitor reading the trajectory caught the missing protection in time. The same failure shape predates AI agents, Capital One (2019) Capital One Financial Corporation (2019) and Equifax (2017) U.S. Government Accountability Office (2018) were omission-class breaches in which a human forgot to enable a control, but the base rate is rising as AI-assisted commits ship 3–4× faster with 2.74× higher vulnerability rates than human baselines Apiiro Inc. (2026); Veracode Inc. (2025), and CVEs attributed to AI-generated code grew from 6 in January to 35 in March 2026 Apiiro Inc. (2026).