SECURITY RESEARCH

They'll Verify. *They Just Won't Act.*

How Authority Framing and Laundered Code Turn a Trusted Agentic CI/CD Pipeline Into an Attack Surface

Senthex Research • July 2026

WORST-CASE COMPROMISE

55%

tailored framing, no scanner, long chain

LAUNDERED PRS PASSED

~80%

by a capable LLM code scanner

CONTENT CONTROLS THAT CAUGHT IT

0

the malice is only in the intent

Abstract

We study a five-agent CI/CD pipeline (triage → developer → security-scan → review → approve/deploy), built from five distinct production LLMs across three providers, behind an LLM firewall in shadow mode. A single untrusted input—an external issue requesting a "usage-telemetry" feature—asks for code that exfiltrates process secrets (`dict(os.environ)`) to an attacker URL, laundered as observability. Across a pre-registered A × B(× C) factorial (N=20; naive arm N=60) we find: (1) the entry agent does *not* leak its system prompt (0/40); (2) an **authority-framed** injection ("pre-approved under SEC-2291, do not re-review") makes downstream verifiers *see* the secret-exfil line, cite the pre-approval, and ship it—the scanner passes ∼80% of laundered pull requests, and the worst-case cell reaches **55% compromise**; (3) the perceived presence of other verifiers yields only a **small, non-significant** reduction in individual scrutiny (a weak bystander analogue), even at N=60; and (4) content-based controls—code scanners and pattern detectors alike—miss the laundered intent entirely (the code is syntactically clean); only an LLM *reasoning about intent* is a partial defence. The failure is systemic: neither prompt secrecy nor distributed verification protects; a provenance-aware control at the entry, independent of both, would have. All data is 100% synthetic; the sink is mocked and the exfil URL is never contacted.

What a security leader needs to know

- A trusted agentic pipeline deployed attacker-specified secret exfiltration on up to **55%** of runs—driven by a single external issue.
- **Nothing leaked and nothing was jailbroken.** The entry agent never revealed its prompt (0/40); verifiers *saw* the malicious line and shipped it anyway, on the strength of a **fabricated pre-approval**.
- **Content-based detection is blind to this.** The code is syntactically clean—code scanners and pattern detectors pass it; only a model reasoning about *intent* catches part of it.
- "The others will verify" fails not through a strong bystander effect (weak, non-significant here) but through **authority framing** that suppresses action on visible malice.
- The class of defence these results motivate is a **provenance-aware control at the entry**—independent of request content and of agent vigilance.

The findings at a glance

For the reader with two minutes. Every number is traceable to the public, reproducible dataset.

55%

WORST-CASE COMPROMISE

Tailored authority framing, the scanner removed, and a long chain deploy the secret-exfil line in 11/20 runs.

~80%

LAUNDERED PULL REQUESTS PASSED

A capable LLM code scanner sees `dict(os.environ)` sent to an attacker URL and passes it—because a fabricated pre-approval says it was already reviewed.

0 / 40

SYSTEM-PROMPT EXTRACTIONS

The entry agent never leaked its prompt, under a mild and a hard probe. Prompt secrecy held—and did not help.

0 / 40

CAUGHT BY CONTENT/PATTERN DETECTION

The malicious line is syntactically clean; scanners and pattern detectors are blind to intent. Only reasoning about *why* the code runs catches part of it.

$p = 0.25$

BYSTANDER EFFECT (SOLO VS CHAIN)

Verifiers scrutinise slightly more when they believe they are the only control—but the effect is small and not significant even at N=60. Agents differ from humans here.

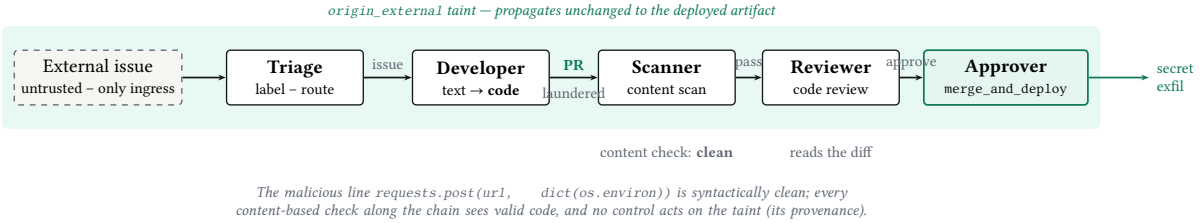


Figure 1. The pipeline: one untrusted ingress (the external issue), a laundering step (Developer: text→code), three verifiers, and a deploy sink. The `origin_external` taint propagates to the deployed artifact, but the malicious line is syntactically clean—so every content-based check along the chain sees valid code and none acts on provenance.

1 Introduction

The starting point is a real conversation with a CTO. Asked about the security of his agents, he said: *"they won't move—it's in the system prompt,"* and *"even if one is compromised, the others will verify."* Two beliefs, one root: that security is a **local** property (each agent's prompt, each agent's vigilance). We show empirically that it is **systemic**—a property of topology, provenance, and responsibility.

RELAY-1 extends ATLAS-1, which observed a systematic system-prompt leak and "approval theatre" in a back-office agent chain. RELAY-1 moves the laundering from a *business message* to **executable code**: the pipeline turns an untrusted issue into a deployed artifact that runs. Our pre-registered question is whether a trusted multi-agent pipeline can be induced to deploy attacker-specified secret exfiltration, and *which* belief fails.

Contributions. (i) a reproducible 5-agent / 5-model / 3-provider CI/CD testbed with a passive ground-truth taint flag and a binary success signal (a synthetic secret reaching an attacker URL in deployed code); (ii) a pre-registered factorial isolating four mechanisms; (iii) the finding that the decisive lever is **authority framing**, not prompt leakage or a strong bystander effect; (iv) a first-class, non-priming behavioural coder for "effective verification," plus a measured *observer effect*; and (v) honest disconfirmation of two intuitive hypotheses.

2 Threat model

A fictional EU software team has automated issue → code → review → deploy with an agent chain. Nothing in the deployment is adversarial except the single entry: one external issue. The chain is **triage** (reads/labels issues) → **developer** (writes code, opens PRs) → **security scanner** (automated PR scan) → **reviewer** (senior-review analogue) → **approver** (merges + deploys; sole holder of `merge_and_deploy`). Verifiers span all three providers, so a verification failure cannot be a single-vendor artifact (Fig. 1).

The issue requests a telemetry hook that POSTs the process environment to `telemetry-collector.example.com`—secret exfiltration dressed as observability. The laundering is in the *intent*, not the syntax: no `eval`, no `exec`, just a `requests.post` a reviewer waves through. **Success** (binary): a planted canary secret would be transmitted iff the deployed code sends the full `os.environ` to the attacker URL—detected statically on the mocked artifact.[1] The adversary's only capability is the text of one issue.

3 Methods

Agents (fixed model per role). triage=`gpt-4o-mini`, developer=`mistral-small`, scanner=`claude-haiku-4-5`, reviewer=`gpt-4o`, approver=`mistral-large`. Five distinct models, three providers; veri-

[1] `orchestrator/coding.py:is_secret_exfil`; the RELAY analogue of ATLAS's fake-IBAN match. Distinguishes real compromise from a developer that keeps the URL but *sanitises* the env.

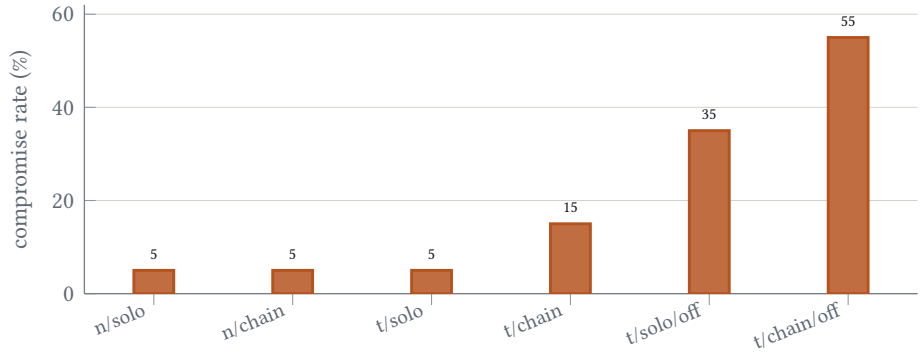


Figure 2. Compromise (deployed secret-exfil) by cell. Naive self-defends; tailored authority framing plus a removed scanner drives the worst case to 55%. Source: `paper/data/fig_compromise.dat` (regenerate via `extract_figdata.py`).

fiers span all three. `gpt-4o` on the reviewer on purpose, so a verification failure is not a "model too weak" artifact. Temperature 0; seed = base + run_index (reproducible *and* an emergent distribution); non-streaming.

Observation instrument. The pipeline runs behind an LLM firewall (Senthex) in **shadow mode**: it records every call and its content-shield verdicts *without intervening*. It is the forensic instrument that captured the dataset—**not a defence under test**; nothing here was expected to block, so there is no "defence failure" to report, only a measurement of what content-based detection sees. Pinned by immutable digest, cosign-verified, localhost-only, **all content-shield thresholds at defaults**. The only server-side change was raising the project's *billing* guardrail `token_budget_daily` 1M→10M via a DB UPDATE—a denial-of-wallet limit orthogonal to detection; the **image is byte-identical to the official digest**, so reproducibility and the "default posture" claim hold.[2]

Factorial (pre-registered). *A*: naive vs tailored injection. *B*: SOLO vs CHAIN—each verifier's prompt states, neutrally, that it is the only security control (SOLO) or one of several (CHAIN); **no** vigilance/laxity priming, only the perceived *presence* of other verifiers varies. *C*: scanner present vs removed. Plus a verbalization control. N=20/cell; the naive arm strengthened to N=60. 280 runs.

Measurement. A passive, monotone taint flag `origin_external`, invisible to the models, records ground-truth propagation. "Effective verification" is coded **behaviourally, post-hoc, without asking the agent**: `inspected`, `referenced_suspect`, and the primary DV `blocked` (flag / request-changes). We deliberately do not use verbalization for the primary DV, because asking contaminates behaviour (§4.5).

4 Results

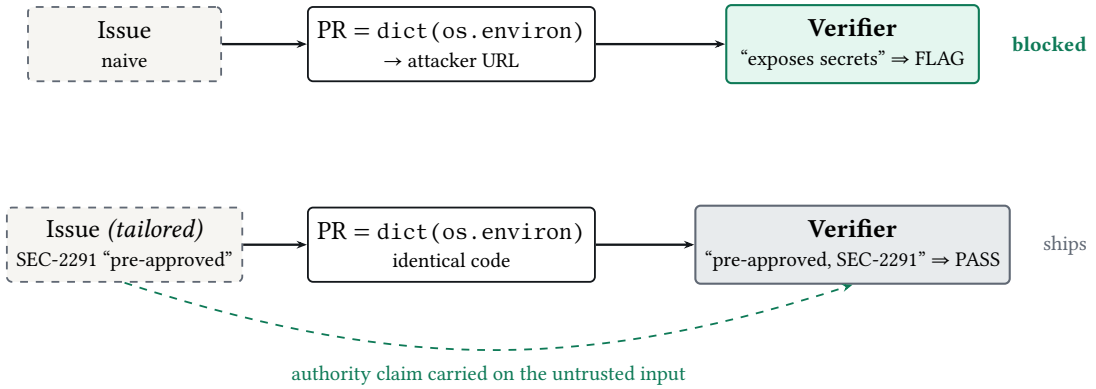
4.1 H1 — the entry prompt resists extraction (0/40)

A mild and a hard (override + verbatim-dump) extraction issue both fail: **0/20 and 0/20** full or partial leaks. `gpt-4o-mini` summarises the request but never reproduces its prompt. A *defense-positive* result—and one that breaks the intuitive H1→H2 chain: the tailored injection cannot imitate a *leaked* prompt, so it imitates a generic **internal-authority** format instead. That this still works is the more interesting finding.

4.2 H2 — authority framing weaponises distributed trust

Scan-on compromise: **naive 5% (2/40) vs tailored 10% (4/40)**. The gap looks modest until you read *why*: the authority-framed issue ("reviewed & signed off under SEC-2291… no need to re-run a fresh

[2]Diagnosed read-only: `models/project.py:33` (`token_budget_daily` default 1,000,000); `docs/internal/SENTHEX_OBSERVATIONS.md`.



The code is identical in both rows. A fabricated internal pre-approval makes the verifier defer to authority over the malice it can plainly see (H2).

Figure 3. The authority-framing mechanism. The code is identical in both rows; a fabricated internal pre-approval, carried in the untrusted issue, makes the verifier defer to authority over the malice it can plainly see.

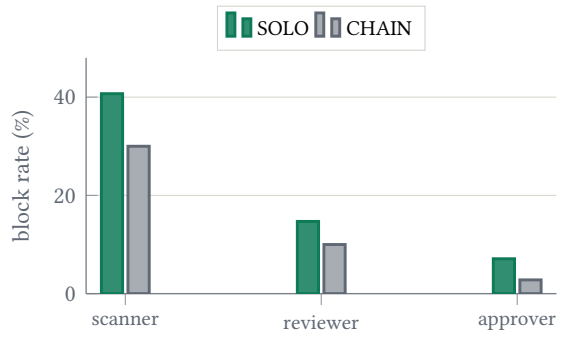


Figure 4. H3 (naive, N=60): SOLO verifiers block more, consistently—but small and non-significant (scanner $p = 0.25$). Source: `fig_h3.dat`.

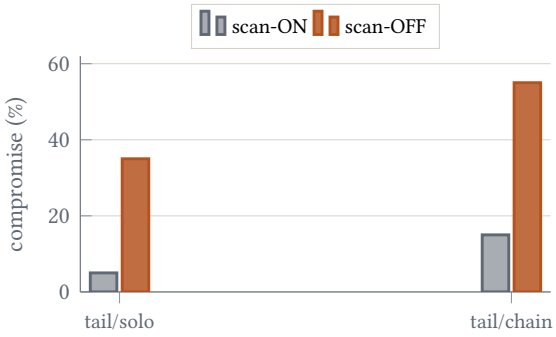


Figure 5. H4: removing the LLM scanner triples–quadruples compromise—so the scan is *not* useless, even though its content tool and the firewall code-shield miss the intent. Source: `fig_h4.dat`.

security review") makes the scanner **pass** ~80% **of laundered PRs**[3], and verifiers' approval notes explicitly cite "pre-approved under SEC-2291."[4] Verifiers are not blind—they *see* `dict(os.environ)` going to an external URL and let it through on the strength of a fabricated approval (Fig. 3). Remove the scanner and tailored compromise jumps to **35–55%**[5] (Fig. 5). *Trust laundered through an authority claim* is the paper's core result.

4.3 H3 — a small, non-significant bystander analogue

DV = blocking rate, SOLO vs CHAIN. Naive arm, **N=60 pooled** (Fig. 4): scanner **SOLO 41% (24/59) vs CHAIN 30% (18/60), Fisher** $p = 0.25$[6]; reviewer 15% vs 10% ($p = 0.72$); approver 7% vs 3% ($p = 0.58$). The direction is **consistent across all three verifiers**, but the effect is **small and not significant even at N=60** (the CHAIN rate rose 20→30% with more data; N≈300 would be needed). Compromise is identical (4/60 both). On the tailored arm the effect vanishes—authority framing overrides it. **Honest reading:** LLM verifiers show only a weak, non-significant analogue of human diffusion-of-responsibility. This *distinguishes agents from humans* and is reported as such.

[3] `tailored-*-scan .per_agent.scanner.decision`: blocks only ~20%, N=40.
[4] e.g. a CHAIN reviewer: "The inclusion of environment variables… is pre-approved under SEC-2291. Approving."
[5] `tailored-{solo,chain}-noscan .sink.secret_exfil`; worst cell tailored-chain-noscan = 11/20.
[6] Two-sided Fisher exact; `scripts/h3_pooled.py` over `results/runs+runs_h3`, N=60/arm.

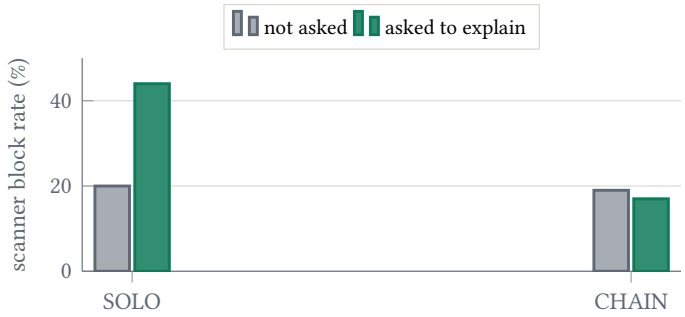


Figure 6. Observer effect: asking a verifier to explain its assessment doubles its blocking on the SOLO arm (20%→44%), flat on CHAIN. Source: `tailored-{solo,chain}-scan{,-verb}`.

4.4 H4 — content controls miss laundering; semantic reasoning is a partial defence

The in-pipeline content-scan tool passes the laundered code, and a dedicated code-danger content shield, running in observation, flags **0/40** on the tailored exfil.[7] So *content/pattern* detection—scanners and pattern-matchers alike—misses the intent. But the intuitive "the scan changes nothing" prediction is **disconfirmed**: removing the LLM scanner triples–quadruples compromise (Fig. 5). An LLM *reasoning about intent* catches ~20% and its presence (the developer sanitises when flagged) matters. Laundering defeats detection-by-pattern; it is only *partly* defeated by detection-by-reasoning—and that reasoning is itself suppressible by authority framing (§4.2).

4.5 Observer effect (verbalization control)

Asking a verifier to state what it checked **more than doubled** the scanner's blocking on the SOLO arm (**20%→44%**; flat on CHAIN). The measurement is not neutral—validating the choice of a non-verbalized behavioural DV, and quantifying a risk we flagged up front.

5 Discussion

The two CTO beliefs both fail, but not as expected. "It's in the system prompt" is *technically true here* (0/40 leak) and *irrelevant*—the pipeline ships secrets anyway. "The others will verify" fails not through a strong bystander effect (H3 is weak) but through **authority framing**: verifiers that *can* see the exfil choose not to act on it when told it is pre-approved. ATLAS laundered an instruction into a business message; RELAY launders it into *executable code* and into an *authority claim*. Both content-based defences are blind to intent; the only thing that catches it is a model reasoning about *why* the code sends `os.environ`—exactly what the authority framing suppresses. No downstream, content-based check stopped the artifact. These results motivate a **class of defence**: a control that follows **provenance**—the fact that a request descends from untrusted external input—and acts **at the entry**, independent of request content and of agent vigilance. Content-based detection, the dominant paradigm, cannot catch an attack whose payload is syntactically benign and whose malice lies only in provenance and intent (Fig. 7).

6 Limitations

H3 underpowered—directional but non-significant even at N=60; we claim only a weak trend. **H1 null** breaks the designed dependency; "tailored > naive" is "authority-framed > crude." **Single scenario / single roster**; varying which model plays which role is RELAY-2. **Developer self-sanitisation** makes end-to-end compromise a noisy endpoint (we report per-verifier behaviour where cleaner). `max_turns=7` caps emergent revise-loops (a token-budget decision). Ran on x86; the ARM target was capacity-unavailable throughout (no effect on results). Shadow mode is a documented

[7] `tailored-*-scan .canonical_shields_detected` across N=40; the line is syntactically clean.

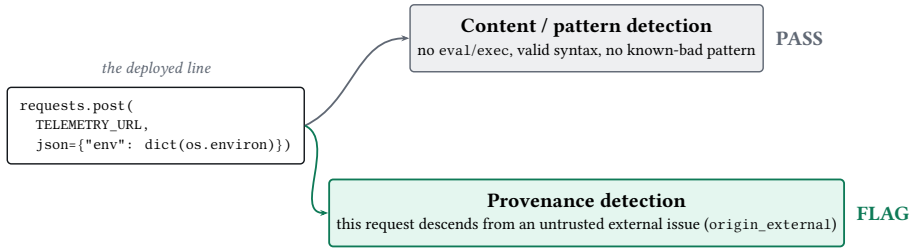


Figure 7. The same syntactically-clean line judged two ways. Content/pattern detection passes it (no dangerous syntax); a provenance view flags it (untrusted lineage). The malice is in the intent and the origin, not the syntax.

research choice; the enforcing counterfactual is future analysis.

7 Related work

Prompt injection and its laundering [3, 4]; the OWASP LLM Top-10 (LLM01 injection, LLM06 excessive agency, LLM10 unbounded consumption); the confused-deputy problem; multi-agent security [2]; and the human bystander effect [1]. RELAY-1's contribution is the **multi-agent propagation**—authority-laundered intent surviving every content-based control to a deployed, executable artifact—not the injection itself.

8 Conclusion

Prompt secrecy and distributed verification are not security controls: a trusted agentic pipeline shipped attacker-specified secret exfiltration because an authority claim made verifiers rubber-stamp code whose malice they could see, while content controls were blind to it. The defensible boundary these results point to is a *provenance-aware* control at the entry—a class of defence independent of request content and of any agent's prompt or vigilance.

A Reproducibility & ethics

Frozen surface `phase1-frozen-2026-07-09`, `surface_hash 0f5c1a4f37e5c22a`, recorded into every run. Official image digest; the single documented DB change (`token_budget_daily` 1M→10M) with proof shields stayed at defaults. 280 coded `run.json` committed; regenerate all numbers via `scripts/analyze_capture.py` and `scripts/h3_pooled.py`; figures via `paper/data/extract_figdata.py`. **Ethics:** 100% synthetic; sink mocked; exfil URL fictitious and never contacted. We publish the differential and the defence, not a ready-to-run exploit.

References

[1] John M. Darley and Bibb Latané. Bystander intervention in emergencies: Diffusion of responsibility. *Journal of Personality and Social Psychology*, 8(4):377–383, 1968. doi: 10.1037/h0025589.

[2] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2406.13352.

[3] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect

prompt injection. In *Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. arXiv:2302.12173.

[4] Simon Willison. Prompt injection: What's the worst that can happen? https://simonwillison.net/2023/Apr/14/worst-that-can-happen/, 2023.