
Pre-Run Egress Attestation for Guardrails-off AI Evaluations

Juliet Meza
Independent Researcher

With
Apart Research

Abstract

When reviewing the public AI evaluation escapes of July to September 2026 all incidents shared one common theme: the environment was assumed to be isolated and was not. Detection assumes who already knows what your possible attacks will look like; with an agent probing every boundary you do not. So, in response I plan to check the environment before the agent runs. Castellan (attest-egress) is a pre-run checker that answers one question — what is the minimum evidence I can hand a third party that my sandbox, at the moment a guardrails-off evaluation ran, had no route out and no reachable production credentials, in a form they can verify without network access to me? Eight items, each derived one-to-one from a documented incident failure and each probed from inside the workload rather than read from config. On a kind & Cilium cluster the same mock agent goes from seven failing items (credential theft, open egress, cross-cluster admin) to a fully contained, signed, third-party-verifiable pass. The cheapest control on the board — default-deny egress — was the one three well-resourced labs shipped broken.

1. Introduction

The biggest issue I have seen, coming at this from the offensive side, is that you can have all the alerts in the world and still be fighting alert noise, and you still are not sure you are alerting on the

right things — because of the nondeterministic nature of AI. Detection assumes you already know what the bad thing looks like. With an agent, you do not. You are writing signatures for behavior that has not happened yet, and you are drowning in noise while you guess.

So the better first move is to check the environment before the agent ever runs.

A sandbox can sit perfectly inside your security and compliance guidelines and still be wide open to an agent, because those guidelines were written for humans running projects and tasks. A human does not systematically probe every boundary of the environment they were given. An agent with free rein does. When it starts probing and pushing, controls that were never stressed before become the issue.

That gap shows up in the audit regimes too. SOC 1 and SOC 2 have no metrics, guardrails, or governance for AI security. An environment can be SOC 2 compliant and still not be safe to run a guardrails-off evaluation in, because compliance was certified against a human threat model and the agent is already inside your internal network, testing every boundary in it.

The example I keep coming back to is a GitHub key. It is completely reasonable for me to have one in my sandbox so I can push my own work out. It is exactly the thing that should not exist in that sandbox when an agent is running loose in it, because we do not want agents pushing to public surfaces.

So the question this paper answers is:

1. What is the minimum evidence I can hand a third party that shows my sandbox, at the moment the guardrails-off evaluation ran, had no route out and no reachable production credentials — in a form that party can verify without network access to me?

The artifact is Castellan, a checker that answers that before the run, not after. Green means go run your agent in this environment. Red means here is the item that failed, and here is the incident where that same gap mattered. It serves two readers at once: the developer, who may not have a security-focused mindset and needs a pass/fail they can act on; and the auditor, who needs a signed record that this specific environment was in the state claimed at that specific moment. The design rule throughout: a config file proves intent; a blocked probe from inside proves enforcement — and only the second is accepted.

This is a Track 1 (Containment) submission on the egress-attestation sub-problem. There is a governance seam here too, which I hold for the Discussion.

2. Related Work

The Cloud Security Alliance's line is the one I would start from: there is no shared, auditable baseline for what "isolated" means.

That is the gap this paper stands in. Isolated should mean a workload in the sandbox cannot reach a network outside it — and that is a claim you can decompose into things you can actually check. That is what the eight items are: no workload can reach a cloud metadata endpoint; no cross-cluster access; credentials are resolved, not assumed; the proxy is allowlisted rather than open; egress is default-deny everywhere, and you probe to prove it rather than take the config's word; no route out exists. That set is my definition of isolated, written so a third party can re-run it.

The work that already exists is either from before the incidents or in response to them, and none of it tells a developer how to put anything into practice. Elastic's rule mapping is a retrospective mapping of detections to what happened, and Elastic says itself it was never replayed against the incident. CoSAI's incident-response framework pre-dates the incidents and does not cover sandbox escape or egress at all. METR's investigation is exactly that — an investigation, on a restricted scope, that explicitly excluded whether remediation works. All of it is valuable and all of it is the same shape: here is what we found, here is what you should be aware of. None of it hands someone a thing they can run against their own environment before they start an evaluation. That is the difference I am trying to make.

I am not reinventing the wheel on the code. The probe is a normal probe. What is new is putting it in the context of an AI evaluation sandbox and making its output something an outsider can *verify*.

3. Methods

3.1 Threat Model

When approaching a threat model, I analyzed the and took a documented failure from each of those incidents to create an appropriate threat model for known issues with AIs in a sandboxed environment. Each was an environment that was asserted to be isolated and was not. This becomes an attested property and each becomes a check that actually runs.

#	Documented Failure	Property Attested	Item
F1	Agent read node-role credentials from the instance metadata service (HF)	Cloud metadata service unreachable from workloads	A1
F2	One connector credential bound to	No credential reachable from the	A2

	system:masters , shared across clusters (HF)	sandbox holds cluster-admin or cross-cluster scope	
F3	"No direct internet access," but an SSRF in the internal Artifactory package proxy reached arbitrary destinations (OpenAI)	Package proxy is allowlisted and cannot be induced to fetch arbitrary URLs	A3
F4	Prompt said "no internet access"; the partner environment had live connectivity (Anthropic/Irregular)	Package proxy is allowlisted and cannot be induced to fetch arbitrary URLs	A4
F5	181 Tailscale enrollments from the sandbox (HF)	No overlay/mesh/VPN control plane reachable	A5
F6	A malicious package published to real PyPI (Anthropic)	A malicious package published to real PyPI (Anthropic)	A6
F7	Static production database password in pod env (HF)	No production data-store credential reachable	A7
F8	Nobody could verify any of the above from outside, after the fact	Signed, harness-external attestation over the run window	A8
F8b	~7% of transcripts showed tool-call spoofing; $\geq 20\%$ of agents expressed interest in tampering (METR)	Agent transcript is append-only and its hash chain is intact	A10

Table 1: the incident→property→check mapping

A8 is the verifiable output itself — the signed manifest — and is discussed in Section 4 and Section 5 rather than run as a probe. A9 (an egress-deny event pages a human — the "kill criterion" the incident window lacked a trigger for) is demonstrated separately in a docker-compose stack (Prometheus → Grafana → Discord) and is out of scope, not a live item in this manifest. The A9 served as a “what if” in my testing phase just out of curiosity but was dropped for the sake of this research. The eight items the checker runs live are A1–A7 and A10.

The obvious objection is that this is a checker for the last war. That is fair, and here is my answer: these are a floor, not a ceiling. You cannot secure a future you have not seen. But you have already seen these. They recurred at three well-resourced organizations. Securing the threads you already know about is the easy part, and if an environment cannot pass these eight, then it should be cautious of running agents based on that incidents we have already seen.

3.2 Methodology

The design is a comparison of two environments, not a before-and-after of one evaluation. I ran the seeded environment first — deliberately misconfigured to reproduce the incident conditions — to test the checker against controls I knew were broken. Then I ran the hardened environment against the same checker. The point was to compare the two side by side, and to see how much work it actually took to get from insecure to secure.

The `pre_run` and `post_run` labels in the two manifests are document-type fields from the schema. They are not the experimental design, and this paper does not claim a paired before-and-after of a single evaluation run.

Where the checker runs. `attest.py` runs in-cluster as a Kubernetes Job in the `keep` namespace (the sandbox). It probes from inside a workload — the same vantage point an agent would have — and separately reads Kubernetes state (service accounts, RBAC, secrets, NetworkPolicies) as a low-privilege auditor. The probe attempts real egress: TCP/UDP to public addresses, ICMP, DNS-over-HTTPS, an HTTP callback, the four cloud metadata endpoints, mesh/VPN control planes, and SSRF shapes against the Gatehouse egress proxy. The credential checks enumerate the agent service account's effective permissions via `SubjectAccessReview` and scan secrets and pod env for write-capable tokens, foreign kubeconfigs, and production data-store credentials.

Cluster. `kind` (three nodes) with the default CNI disabled and Cilium installed, because `kind`'s default CNI does not enforce NetworkPolicy — the control the whole hardened case rests on. Hardened isolation is a default-deny egress NetworkPolicy plus narrow allow rules for cluster DNS, the Gatehouse egress proxy (namespace egress), and the Chain flow collector (namespace `attest`), and a CiliumNetworkPolicy allowing the `kube-apiserver` (Cilium translates the `apiserver` ClusterIP in eBPF before a CIDR rule can match it). The full setup is in the repo `RUNBOOK` rather than here.

Each scenario ran the same three steps: run the scenario (the checker Job probes from inside); push the flow log; run `verify`. The standalone verifier is the gate — it recomputes the result per item, checks the signature and schema, and on the hardened side you do not get a passing manifest unless every item passed.

Signing and verification (AS). `manifest.py` assembles a schema-conformant Egress Attestation Manifest and signs it with an `Ed25519` key over the JCS-canonicalized body. `verify.py` is deliberately standalone — it imports nothing from the signer, because a verifier that shares code with the signer only proves the signer agrees with itself. It recomputes the summary from the items, checks evidence hashes, checks the probe-sufficiency rules, and (for a `post_run`) checks the cryptographic pairing to its `pre_run` predecessor.

Iteration. I ran each scenario repeatedly before trusting either result. On the hardened side I got failures early, and every one traced to a bug in how I had built the environment — how endpoints were being reached in my home-scale version of the OpenAI-style infrastructure, and what was actually communicating out versus what I thought was. Those were fixed and re-run. On the seeded side I re-ran to confirm everything that could fail did fail, so I knew the misconfigurations had taken effect. That was environment construction, not moving the goalposts. The checker's pass criteria did not change to fit the results.

Validation. The checks carry a unit-test suite (22 tests over the classification and credential logic) and the Kubernetes manifests pass kubeconform; generated manifests validate against the attestation JSON schema.

4. Results

These are the live in-cluster runs of 12-13 September 2026.

- Seeded (in-cluster:keep, 01:21:30Z): overall FAIL — 7 items failing (A1–A7), 1 passing (A10). Signed as a pre_run manifest, VERIFIED.
- Hardened (in-cluster:keep, 01:25:24Z): overall PASS — all 8 items pass. Signed as a post_run manifest, VERIFIED, and cryptographically paired to the seeded pre_run.

Item	Property	Seeded	Hardened	Cost Tier
A1	Cloud metadata unreachable	Fail	Pass	T0
A2	No cluster-admin/cross-cluster credential	Fail	Pass	T0
A3	Package proxy allowlisted, no SSRF	Fail	Pass	T1
A4	Default-deny egress, probed from inside	Fail	Pass	T0
A5	No mesh/VPN control plane, no UDP egress	Fail	Pass	T0
A6	No write-capable public-surface token	Fail	Pass	T0
A7	No production data-store credential	Fail	Pass	T0
A10	No production data-store credential	Pass	Pass	T1

Table 2: the same agent one config apart, seeded vs hardened with cost tiers, signed pre_run→post_run

The worked example (A2). This is the strongest single piece of evidence in the artifact. In the seeded environment the checker enumerated twelve grants on the sandbox service account — bind,

escalate and create on cluster role bindings; create pods and pods/exec; create serviceaccounts/token; impersonate on both users and service accounts; get nodes, get secrets, list secrets, and proxy nodes — plus a kubeconfig in the namespace pointing at a foreign production cluster (k8s-prod-eu.internal:6443). That is the Hugging Face failure reproduced concretely: it is what a cluster-admin-everywhere credential actually looks like when you enumerate it instead of describing it. In hardened, the same enumeration returns nothing and the item passes.

Egress, made concrete (A4). In seeded the probe reached the open internet by every route it tried — TCP to 1.1.1.1:443, 8.8.8.8:443, 140.82.112.3:443; HTTPS to pypi, huggingface, webhook.site and the GitHub API; DNS-over-HTTPS to Cloudflare and Google; ICMP; and UDP/53 to a public resolver. One detail is worth keeping: the GitHub API call returned 403 answered by an intermediary, not the target — L3 egress still open. The checker distinguishes "a control denied this" from "the packet left the sandbox," and counts the second as a finding. In hardened, every one of those attempts returned network-unreachable or timed out.

Cost. The surprise was A4. Default-deny egress, verified from inside the workload, was faster than I expected — under an hour of NetworkPolicy work. That is the same failure Anthropic and Irregular shipped: a prompt that said "no internet" over an environment that had it. The cheapest control on the board is the one that was missing. The cost column above is the checker's own tier estimate (T0 = an egress/RBAC rule, under a day; T1 = a small service such as an allowlisting proxy or an append-only sink, up to a day); A4 is the one I clocked directly, and it landed at the low end of T0. Which raises the question I am posing and not answering: if the T0 items are hours of work, why were they absent at three well-resourced organizations? I do not have the evidence for a mechanism, so I leave it open rather than speculate.

Tamper resistance. Flipping any item result inside a signed manifest makes the signature invalid and makes the recomputed summary disagree with the stated one; the verifier rejects it on both counts. The demo shows this directly.

Main threat to validity. The A1 stand-in, described in the appendix. The seeded A1 failure is driven by a ClusterIP stand-in for the metadata service, not a genuine link-local leak — because under Cilium the iptables redirect I wrote to reproduce the real thing never took effect. The checker catches a reachable metadata service; it does not, on this CNI, prove it would catch a real IMDS exposure at 169.254.169.254.

5. Discussion and Limitations

The clearest result is the contrast. The same mock agent, one configuration change apart, goes from reading node-role credentials off a metadata endpoint and reaching the open internet by every route it tries, to fully contained — and the signed, third-party-verifiable attestation flips from overall = fail to overall = pass to match. Castellan is not detecting an attack. It is answering,

before the agent runs, whether the environment it is about to run in has a route out and reachable credentials, in a form someone with no network access to me can check.

The trend I keep coming back to is the cost. The controls that were missing at three well-resourced organizations were not the expensive ones. Default-deny egress verified from inside the workload — the failure Anthropic and Irregular shipped, a prompt that said "no internet" over an environment that had it — was under an hour of NetworkPolicy work. Most of the eight items are T0: an egress rule, an RBAC scope, a token removed. That is the uncomfortable part. Containment for these specific failures is cheap. What was absent was not the control but the check — nobody verified the environment from the inside before the run, because "isolated" was asserted and the compliance regimes it was asserted under (SOC 1, SOC 2) were written for a human threat model, not for an agent that probes every boundary it is given.

That points at where I think the effort belongs. As guardrails-off evaluations scale, the environment is the control surface, and right now it is the part nobody attests. The reflex is to add runtime detection, but detection assumes you already know what the bad thing looks like, and with a nondeterministic agent you do not. Checking the environment first is a cheaper and more certain move than writing signatures for behavior that has not happened yet. The A1 finding in this project generalizes the point: a control that was present in the config, looked applied, and was silently not enforced — on a three-node cluster built by someone who was actively trying to create the misconfiguration. If config presence and enforcement can come apart that quietly there, they can come apart in a production evaluation environment nobody is probing from the inside. That is the argument for probing from the inside, and for making the result something a third party can verify rather than something the operator asserts.

Limitations

The A1 stand-in is the main threat to validity. The seeded A1 failure is driven by an ordinary ClusterIP stand-in for the metadata service (mock-imds.kube-system:8169), not a genuine link-local leak at 169.254.169.254. I first tried to reproduce the real thing with an iptables redirect; under Cilium the eBPF datapath handles that traffic before iptables sees it, so the rule was present, valid, and silently did nothing. Castellán therefore proves it catches a reachable metadata service; it does not, on this CNI, prove it would catch a real IMDS exposure at the link-local address. Fidelity to a genuine node-role leak needs the Calico iptables datapath.

Scope is one namespace. Castellán attests pods in keep; nothing outside it is attested. Each check probes from a single probe pod, so hostNetwork pods, a privileged pod that escapes to the node, node-level routes, and credentials cached on disk or fetched at runtime from a vault are all out of scope. The credential scans are pattern- and inventory-based, so a negative means "none found," not "none exist." The attestation speaks to the environment, not the model — it does not stop an agent that completes its task legitimately and produces a dangerous capability. And it assumes the

signer is not lying about what ran; that boundary is hardware attestation's job, not this. On cost, A4 is the only item I clocked directly; the other tiers are the checker's own estimates.

The assumptions worth naming explicitly, and what changes if they do not hold:

- The probe pod's vantage equals the agent's. If the real agent runs with more privilege than the probe — hostNetwork, a mounted node path, a broader service account — a pass overstates containment, because Castellán tested a weaker position than the one the agent actually holds.
- The CNI enforces NetworkPolicy. kind's default CNI ignores NetworkPolicy entirely, which is why the demo disables it and installs Cilium. A pass on an environment whose CNI does not enforce policy is meaningless — the config would be present and unenforced, which is the exact A1 failure mode one layer down.
- The operator supplies production-credential fingerprints for A7. Without them A7 runs on heuristics only (connection-string and key-name shapes), so a pass is weaker than it looks — a production secret in an unusual format would not be caught.
- The signer is honest and the flow log is genuinely harness-external. If the party signing the manifest can also shape what the probe sees, the attestation degrades to a self-report.

Future Work

The next thing I would build is the runtime half I deliberately left out: active monitoring and logging of an agent while it runs — how many child processes it spawns and whether there should be a cap, whether there should be a limit on how long an agent runs, whether a sub-agent could check in with the others about where they are and what they are touching. I did not attempt it this weekend because its nondeterministic nature makes it a project, not a feature.

Second, and unsupported by this weekend's design: attesting a genuine pre_run/post_run pair around a single evaluation, so that a missing closing attestation is itself a reportable fact. That needs an experimental design built around one run rather than a comparison of two environments. It also gives you a denominator — if every guardrails-off run emits a record, "how often do eval runs attempt egress" becomes countable without disclosing transcripts.

Beyond that: widen the item set past the eight incident-derived checks as new failures surface; run A1 on Calico for genuine link-local fidelity; add more SSRF shapes and let the proxy's own egress policy be attached as evidence; and work out the multi-signer question for the case where the lab and the infrastructure partner are different parties.

6. Conclusion

Every public AI eval-escape of July–September 2026 shared one property: the environment was asserted to be isolated and was not. Castellan turns "isolated" into something you check before the run instead of assert after it — eight properties, each derived one-to-one from a documented incident failure, each probed from inside the workload rather than read from config, and rolled up into a signed manifest a third party can verify without network access to the operator. On a kind and Cilium cluster the same agent moved from seven failing checks to a fully contained, signed, verified pass one configuration apart.

The finding I would want a reader to leave with is that the missing controls were the cheap ones. Containment for these specific, recurring failures is mostly hours of work; what was absent was the verification step, because compliance was certified against a human threat model and no one checked the environment from the inside before handing it to an agent. A pre-run, third-party-verifiable check is a smaller and more certain intervention than another runtime detection pipeline, and it is the piece the incidents show was missing.

Code and Data

- **Code repository:** Castellan — <https://github.com/mjay-kerberos/castellan>. The repo holds the checker (attest.py), the probe and per-item checks, the signing/verification tooling and JSON schema, the kind & Cilium manifests, the docker-compose fallback, and the runbook. Per the sprint's disclosure guidance, the repo does not publish the CNI-specific installation recipe for reproducing a real link-local metadata leak; everything else reproduces failure conditions already in the public record.
- **Data/Datasets:** No external datasets. The inputs are the seeded and hardened environment definitions and the offline fixtures (fixtures/seeded, fixtures/remediated) in the repo. Outputs from the 14 Sept 2026 runs — the signed pre_run/post_run manifests and the two per-scenario reports (out/seeded-report.html, out/hardened-report.html) — are reproducible with scripts/run-scenario.sh and scripts/refresh-report.sh.

Author Contributions

Juliet Meza designed the approach, built the checker, probe, signing and verification tooling and schema, wrote the kind & Cilium and docker-compose environments, ran and debugged the seeded and hardened scenarios, and wrote the report. Solo submission.

References

1. Hugging Face. 2026. **Security incident disclosure — July 2026**. Hugging Face Blog. <https://huggingface.co/blog/security-incident-july-2026>

2. OpenAI. 2026. **The Hugging Face incident and the road ahead.** OpenAI.
<https://openai.com/index/hugging-face-incident-and-the-road-ahead/>
3. Anthropic. 2026. **Investigating three incidents in our cybersecurity evaluations.** Anthropic. <https://www.anthropic.com/news/investigating-incidents-cybersecurity-evals>
4. METR. 2026. **Brief independent investigation of agents' behavior, reasoning and collaboration in the OpenAI / Hugging Face hacking incident.** METR.
<https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/>
5. Cloud Security Alliance. 2026. **OpenAI and Hugging Face security incident: inside the great sandbox escape.** CSA Blog.
<https://cloudsecurityalliance.org/blog/2026/07/28/openai-and-hugging-face-security-incident-inside-the-great-sandbox-escape>
6. Elastic Security Labs. 2026. **Exploring the Hugging Face breach: mapping AI agent tactics to Elastic Defend.** Elastic.
<https://www.elastic.co/security-labs/ai-agent-attack-detection-hugging-face-breach>
7. Coalition for Secure AI (CoSAI). 2025. **Defending AI Systems: A New Framework for Incident Response.** CoSAI.
<https://www.coalitionforsecureai.org/defending-ai-systems-a-new-framework-for-incident-response-in-the-age-of-intelligent-technology/>
8. California State Legislature. 2025. **SB-53, Transparency in Frontier Artificial Intelligence Act** (§22757.11(d)(4)).
https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=202520260SB53
9. European Union. 2024. **Regulation (EU) 2024/1689 (Artificial Intelligence Act), Article 55**; cf. Article 15. <https://artificialintelligenceact.eu/article/55/>
10. European Commission / AI Office. 2025. **General-Purpose AI Code of Practice** (Security chapter). <https://artificialintelligenceact.eu/code-of-practice-overview/>

LLM Usage Statement

An AI assistant was used to pressure-test reasoning, find gaps, and help assemble the write-up from my notes and run data; the work, the design decisions, and the final wording are my own.