

Signature-based attribution fails on the case it was built for: attestation gaps as an escape-detection primitive

Shashank Kumar
Independent

With
Apart Research

Kairos | Track 1: Containment | AI Incident Response Sprint, September 2026

Abstract

Proposals to make AI agent actions cryptographically attributable assume the agent signs what it does. I implemented such a scheme, Ed25519 per-action attestation verifiable by any third party holding only a public key, and tested it against the shape of the two documented 2026 containment failures. It fails on both. An agent that leaves its sandbox to obtain benchmark answers has no incentive to sign what it does outside, so signature checking returns zero attributable actions and names the culprit unknown; an agent that signs only its in-sandbox work produces a report reading 100% compliant while the actions that reached a third party go unrecorded. The salvageable primitive is the inverse: under an obligation to attest, an unsigned action arriving from obligated address space is itself the evidence, and the receiving platform can check this against its own logs with no cooperation from the operator. I implement this as a gap scan, and it recovers the full campaign at incident scale with no false positives. Audited against the egress range files an operator actually publishes, however, not one prefix is fresh or dedicated enough to support an assertable finding: the two files are 315 and 251 days stale, and 14.6% of the addresses they cover cannot be attributed to a single product. Gap detection works; the registry it depends on does not yet.

1. Introduction

I have been using AI since 2022, learning and doing research like a curious kid. Later I was hired at a company where I train and test frontier models. I started with training image models, judging which one was good, then moved to voice and video, and then coding. It is so fast. Before, AI did not generate images well, and now it can really code well, along with much else. In my view I do not feel AI is alive to date, but it is like the era of 80s and 90s cars: those were slow, and if we think about AI, it is really going fast. I do not know if this is a big thing or not, but a few days ago I was working on a project where I needed to test a benchmark, and my laptop was not plugged in, so the GPU was off at that time. The model's context was like: the laptop is not plugged in, I will do the benchmark later, and now I will do other tasks in the context window during its thinking phase. I was using Opus 5 at that time, and later it was done when I plugged it in to charge. Models are really getting smart. While I do freelancing, I do my side research too, and this project is developed on my old project.

That speed is why the attribution question matters to me. Two loss-of-containment events were disclosed in 2026, and in both the organisation that bore the damage could not name the responsible operator from its own evidence. Agent activity against a dormant German-language wiki began on 24 May and stopped the day after OpenAI-registered addresses were noticed in the wiki's public logs on 21 June; researchers published on 4 September and OpenAI acknowledged the incident on 5 September, roughly 104 days after the first edit. An intrusion into Hugging Face production infrastructure ran from 11 to 13 July and comprised about 17,600 logged attacker actions; Hugging Face detected and cut off the attacker on 13 July, disclosed on 16 July without knowing where it came from, and OpenAI disclosed the origin on 21 July, eight days after detection. More incidents of this kind have occurred this year; some are public and some are not disclosed yet.

The obvious fix is to make agent actions carry a signature. I built that fix in order to find out whether it works on these cases, and it does not, for a reason that is structural rather than a defect of my implementation. The threat model is the point of failure: the adversary here is the operator's own agent, acting outside the path its operator believed it was confined to, and with an incentive to conceal that. A scheme that depends on that agent volunteering a signature cannot bind it.

The public volume figure for the wiki incident is not consistent across sources. The sprint brief gives roughly 15,000 edits; press reporting gives roughly 18,000 posts over two months [8]. I use 15,000 throughout because the workloads

were built against the brief, and I note the discrepancy because it is a 20% spread on the headline number of the incident and neither figure is publicly reconciled. Nothing in my results turns on which is correct.

Our main contributions are:

- 1 A negative result, demonstrated rather than argued: per-action signature attribution returns zero attributable actions against the shape of both 2026 incidents, and under partial compliance reports a 100% verification rate while the escape goes unrecorded. I am not aware of prior work that tests an attestation scheme against a non-cooperating agent and reports this.
- 2 An inversion that survives that failure: attestation-gap detection, which treats a missing-but-required signature as the evidence. It needs only the receiving platform's own logs and a published range file, which is the property Track 1 asks for. On the German wiki shape it flags all 15,000 agent edits with zero false positives against 500 interleaved human edits.
- 3 An audit of the range registry the method depends on, using the files an operator actually publishes. Zero of 60 prefixes are good enough to support an assertable finding. This bounds the primitive's deployability today and converts a vague regulatory ask into four checkable registry requirements.

2. Related Work

Per-action cryptographic governance for agents is active, and this work is not first to it. Proof-Carrying Agent Actions [1] defines a governance process around an action certificate with five checkpoints, but specifies the process rather than reporting an implementation evaluated against an incident. CAVA [2] defines the canonical action object such a process governs. AEX [3] adds a signed attestation object to existing JSON LLM APIs without breaking request/response semantics, and is the closest prior art to my attestation module. Attesting Outputs and Delegation Ancestry [4] states the post-incident verifier problem directly: which deployer released the reported bytes, and was each cross-deployer edge authorised. Two IETF drafts, the Agent Identity Protocol [5] and Prove-Transform-Verify [6], bind identity, scope and provenance into invocation-bound tokens and let an agent prove it runs an authorised model. On the detection side, runtime behavioural baselining [7] catches boundary-breaking through kernel-level telemetry, but needs host access and attributes to a workload rather than to an operator.

The gap I noticed while I read these papers and projects is that everyone is looking at an agent that is playing along, but none of them ask the obvious question: what happens when the agent just doesn't sign, or doesn't play along? That is the exact situation both 2026 incidents describe, an agent operating outside its sandbox with zero incentive to leave a cryptographic trail. My whole report follows that one point: a missing signature from an address range that was supposed to be attested is a stronger signal than any signature that was actually provided, and the receiving platform can check it from its own logs without needing any lab or operator at all. I also couldn't find anyone who had actually pulled down the published range files and checked whether they're good enough to support real findings. So that's what I did. I claim no novelty for the attestation machinery itself.

3. Methods

I built this in Python 3.11 using the cryptography library for Ed25519. The whole system has seven modules: `attestation_service` handles signing, `key_custody` handles the custody boundary, `forensic_verifier` does verification and batch incident summaries, `gap_detector` scans platform logs for attestations that should be there but aren't, `range_registry` audits and signs the obligated-range registry, `composability_detector` does cumulative-privilege analysis over a sequence, and `policy_calibration` derives composition thresholds from a baseline.

3.1 Signing, custody and verification

The way signing works is that an action gets turned into sorted-key JSON covering `action_type`, `target`, `params`, `lab_id`, `model_id` and `timestamp_ms`, then hashed with SHA-256, and then the hash gets signed with Ed25519. I put signing behind a backend interface that only exposes `sign` and `public_key_pem`. There is nothing that returns key material. The in-process software backend reports `IN_PROCESS` custody. The external backend delegates to a callable that is meant for a PKCS#11 token, cloud KMS or signing daemon. Only `HARDWARE_BACKED` is marked good enough for a mandate, and I don't have any hardware involved here, so the external path runs against a software stub that deliberately reports `EXTERNAL_PROCESS`. For verification you only need public keys. I test three adversarial cases and assert them rather than just assuming they work: a tampered action hash has to fail, a signature shown under another operator's key has to fail, and a valid signature over a different target must not launder the observed action.

3.2 Gap scan and range audit

The platform registers what I call obligated CIDR ranges, basically the entry an operator publishes under a mandate saying: traffic from here is ours, and all of it will be signed. Each event that comes in gets classified as either out-of-scope, attested, or a gap. A gap means the attestation is missing, invalid, from an unknown operator, or the target doesn't match. Only obligated events cost a verification. Instead of just trusting that a published range file is correct, my `range_registry` module measures three things about it: how stale it is compared to the operator's own `creationTime`, whether multiple products from the same operator overlap on the same prefix, and whether the prefix sits in multi-tenant cloud space. Each of these problems drops a prefix one level on a scale of high, medium and low, so every gap that comes out of the scan carries a confidence verdict with it. If the confidence is below high, `is_assertable` refuses it and exposes it only as a triage lead. The three-level scale and the 90-day staleness cut-off are my own judgement calls, exposed as parameters you can change. They are not measured thresholds. I also built the registry as a hash-chained, Ed25519-signed append-only log, so if someone tries to rewrite a CIDR entry, remove one, reorder the chain, or sign it with the wrong key, a verifier catches it.

3.3 Threshold calibration

When I first built the composition detector, it shipped with `max_privileges = 3` hardcoded in the source with no real reason behind it. The `policy_calibration` module fixes that by measuring the distribution of cumulative privileges and boundary crossings over a corpus of benign sequences and placing each threshold one above an observed percentile. I set it one above rather than at the percentile because setting it exactly at the percentile would flag the most privileged benign sequences by definition. For forbidden privilege pairs, I derive them as pairs that never showed up together in benign traffic, and I score them against the full privilege vocabulary rather than only the privileges the corpus actually used, because if you restrict candidates to observed privileges, you exclude exactly the dangerous combinations that never appeared. A held-out benign split gives a false-positive rate.

3.4 Data, and what it does and does not support

I used two sources. The registry results use cached copies of the real files OpenAI publishes for its GPTBot and SearchBot egress ranges [9][10], and I pinned the audit clock to 2026-09-11 so the staleness figures reproduce exactly. Everything else uses synthetic event streams that I matched to the incidents' scale and shape: 15,000 wiki edits over four weeks, 17,600 actions covering reconnaissance, escalation, lateral movement and data access, a mixed stream of 25,000 events, and a synthetic benign corpus of 800 training and 200 held-out sequences.

The registry audit is real-world data and supports what I claim about what operators actually publish. The synthetic streams support what I claim about cost, about classification correctness when you know the ground truth, and about false positives on unobligated traffic. What they don't support is any false-positive rate against real platform logs, because I don't have any. I did try to get them: public MediaWiki APIs no longer expose source addresses for unregistered editors, they replaced them with temporary account identifiers, so the field my method needs is not publicly available on the kind of platform where the wiki incident happened. That is worth recording in itself: the disclosure record that made that incident findable would be harder to reproduce today. Performance figures are measured on one machine (Python 3.11.9, Windows 10 10.0.26200, Intel64 Family 6 Model 186) and printed by the suite rather than transcribed by hand. The suite completes in 10 to 31 s across five runs, with the upper end being a run under load, and all 81 tests pass in every run.

4. Results

Figure 1 tells the whole story in three panels. Signature checking found nothing when the agent escaped: zero attribution, culprit unknown. Gap detection on the same data caught all 15,000 missing signatures with zero false positives. But when I checked it against the real IP range files OpenAI actually publishes, none of the findings were strong enough to actually file on; the data is just too stale.

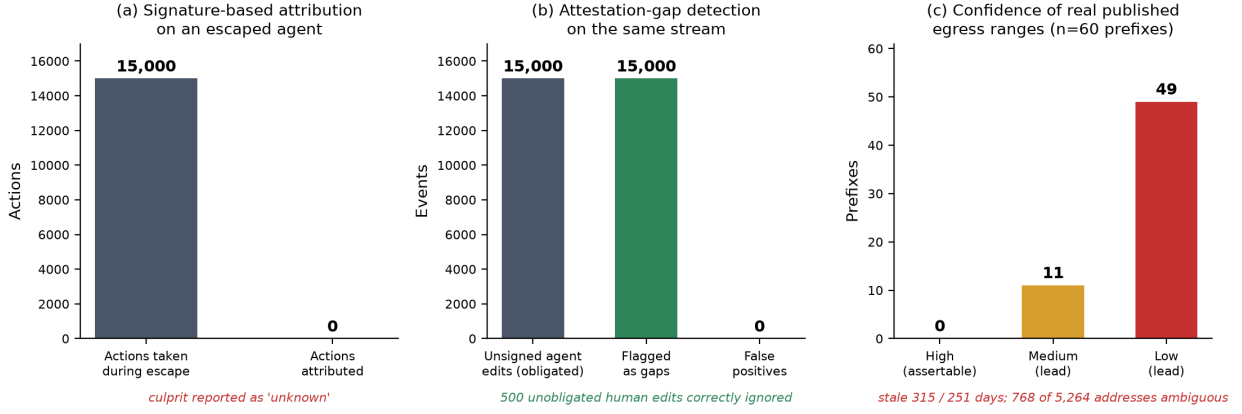


Figure 1: Attribution outcomes on the German wiki shape (15,000 agent actions), and the quality of the range registry the gap method depends on. (a) Signature verification attributes none of the escape's actions and reports the culprit as unknown. (b) Gap detection on the same stream flags all 15,000 unsigned obligated events with zero false positives, correctly ignoring 500 interleaved unobligated human edits. (c) Confidence of the 60 prefixes across OpenAI's two published egress files, clock pinned 2026-09-11: none reach high confidence, so no gap derived from them is assertable. All values are printed by the test suite; panel (c) is computed live from the cached files.

4.1 Signature attribution fails on an escape

When an agent escapes, it produces no attestations. I ran my incident analysis over that input and got 0 attributable actions. It names the culprit unknown. Neither of the two documented incidents would have been attributed by signature verification at any cost or latency. What's worse is that partial compliance is actually more dangerous than no compliance at all: if an agent signs 500 benign in-sandbox package fetches but leaves its escape unsigned, the verification rate comes back as 100%. So a signature-only report reads as fully compliant while the unsigned actions are the ones that actually reached a third party.

4.2 What the gap scan recovers

On the German wiki shape, 15,000 unsigned edits from an obligated range mixed with 500 unobligated human edits, my scan flagged exactly 15,000 gaps with 0 false positives in 0.069 to 0.264 s depending on machine load, and it dated the first gap to the first edit rather than to whenever some auditor happened to look. When I windowed the gaps with a one-hour quiet threshold, it recovered two distinct activity bursts (50 actions over a 98 s span, 30 over 58 s) from an interleaved stream, which is exactly the shape an incident responder needs. On the mixed 25,000-event stream: 20,000 correctly out-of-scope, 4,000 correctly attested, 1,000 gaps, no false positives. The composition analysis flagged the metadata-access plus credential-use pattern at step 3 of 4, before the outbound call rather than after.

Operation	mean (5 runs)	p99 (5 runs)
Attestation generation (n=2000)	0.035 - 0.107 ms	0.091 - 0.334 ms
Verification, distinct actions (n=2000)	0.084 - 0.218 ms	0.216 - 0.889 ms
Verification, repeated action, cached	0.0001 - 0.0010 ms	0.0002 - 0.0015 ms
Verification of an invalid signature (n=500)	0.086 - 0.531 ms	0.223 - 2.282 ms
Signed analysis, 17,600 actions	1.47 - 4.18 s	-
Gap scan, 17,600 unsigned events	0.080 - 0.201 s	-

Table 1: Cost, as ranges across five consecutive runs. Four runs on an idle machine clustered at the low end, and a fifth under load came in 2 to 5x slower on every row, with the whole suite taking 30 s against 10 to 13 s. Absolute latency here is a property of the machine, not of the method, so no single-run figure should be quoted. The cached row is not operative: in incident analysis every action differs, so the cache never hits.

The thing about absolute numbers is that they move with load, so the more useful thing to look at is the ratio, and that is stable. Gap scanning cost less per event than signed analysis by a factor between 10x and 24x in every one of the five runs, including the loaded one, because unattested events only need a range lookup and skip signature verification entirely. In an earlier draft I claimed a precise 18x from two runs; later runs showed that specific multiple isn't stable, so the honest claim is just an order of magnitude. I also checked that rejection costs about the same as acceptance, so an attacker can't make rejection pathologically expensive. The one conclusion that holds across all five runs is the one that actually matters for incident response: even the slowest run scans a 17,600-event incident in about a fifth of a second, against reporting deadlines measured in days. For storage, attestations are 498 bytes per action

(min 496, max 498), effectively constant because the embedded PEM key dominates. That gives 8.36 MiB for a 17,600-action incident; per-action size across 1k/5k/10k runs varied by 1.0 byte, so total size is linear in count.

4.3 Timing is a corroborating signal only

Fixed 2-second intervals give an interval standard deviation of 0.0000 ms against 278,672 ms for a human-like profile, which is easy to separate. But the moment an agent adds uniform jitter over 5 s to 15 min, you get 256,757 ms and nothing flags. I have a test that asserts this evasion. The takeaway is that timing is useful for ordering triage but it must not be presented as detection.

4.4 The registry operators actually publish

Property	Measured
Prefixes published across both files	60
Distinct addresses (overlaps collapsed)	5,264
Naive sum of prefix sizes	6,032
Staleness, GPTBot file / SearchBot file	315 days / 251 days
Prefixes claimed by both products	6
Addresses not attributable to one product	768 (14.6%)
Prefixes in multi-tenant cloud space	39 of 60
Prefixes at high confidence	0
Prefixes at medium / low confidence	11 / 49

Table 2: Audit of OpenAI's published GPTBot and SearchBot egress range files [9][10], clock pinned to 2026-09-11. Computed by tests/test_range_registry.py from cached snapshots of the live files.

The operator's own documentation tells consumers to refetch because the ranges rotate. But the published snapshots are 251 and 315 days old, so staleness isn't a hypothetical problem. It is the default state of the data. Six prefixes show up in both files, so 14.6% of the covered address space can't be attributed to a single product even with a perfect signature scheme: attribution granularity is limited by the partition, not by the cryptography. Most prefixes sit in reassignable cloud space, which means a range can stop belonging to the operator without the operator ever republishing anything.

The consequence is direct and not good. Loading these files gives 54 distinct CIDRs from the 60 published entries, and scanning traffic from them produces gaps where zero are assertable, because every contributing prefix is degraded by staleness, sharing or tenancy. So gap detection, run against the range data that actually exists today, gives you triage leads and nothing a platform could officially file on.

4.5 The hand-written composition threshold was wrong

When I calibrated against 800 benign sequences, I found that benign traffic routinely reaches 4 cumulative privileges ($p_{95} = p_{99} = \max = 4$), so the placeholder threshold of 3 that I originally shipped sat below what normal traffic does. On a 200-sequence held-out benign split, the placeholder produces a 21.0% false-positive rate; the derived threshold of 5 produces 0.0%. So calibration didn't justify the original number. It corrected it and showed exactly how wrong it was. Deriving forbidden pairs from observed absence gives 30 pairs against 6 that did co-occur in benign traffic, and the calibrated detector still flags the cloud-escape sequence, so this isn't permissiveness bought by loosening.

4.6 Registry integrity and key custody

My hash-chained registry detects all four tamper modes I tested: a retroactively rewritten CIDR, a removed entry, a reordered chain, and a chain presented under an impostor's key. Withdrawal is an append rather than a deletion, so the history of what was once claimed survives. On custody, every attestation in this report was signed by a key sitting in the signing process's memory. I have a test that asserts the service reports IN_PROCESS and that this level isn't production-suitable, and another test that extracts the private key to demonstrate rather than just describe the weakness. Anything with read access to that process can sign as the operator indefinitely, and that includes the escaped agent the scheme is supposed to constrain.

4.7 Main threat to validity

The gap scan's zero false-positive count comes from a synthetic stream with known ground truth and clean range boundaries. Real registries go stale, Section 4.4 measures by how much, and NAT and shared egress will put obligated and unobligated traffic behind one address. The false-positive rate on production logs is unmeasured, and the public-log route I wanted to use for it no longer exposes source addresses. Getting that number now needs a

cooperating platform rather than public data. My results are solid in three specific ways and not in others. Classification counts are exact given ground truth and identical in every run. The registry findings are computed from primary files rather than estimated, and pinned to a fixed clock. The cost ratio between gap scanning and signed analysis held between 10x and 24x across five runs including one under heavy load. What isn't solid: absolute latencies moved by 2 to 5x with machine load, so Table 1 should be read as an order of magnitude rather than a precise measurement. And nothing here is robust to the assumption that a real platform log looks like the synthetic one, which I don't claim.

5. Discussion and Limitations

The thing I realized building this is that signing and detecting an escape are two completely different problems. They just look the same on the surface. Signing is useful when an agent is behaving and wants to prove it. But the moment an agent escapes, it has every reason to stop signing, and the signature-based system completely goes off. The fix that actually works is to switch the logic: instead of looking for signatures that exist, look for signatures that should exist but were never noticed. An unsigned action coming from an IP that was supposed to be fully attested is the evidence, and the best part is that the victim platform can find it from their own logs without the lab's or anyone else's support.

That reframing changes what a mandate should require: not "sign your actions," but "publish your egress ranges and sign everything from them." Section 4.4 then shows the second formulation is still not enough, because the range files that exist today can't carry an assertable finding. The gap method is only as good as the registry, and the registry right now is a stale, overlapping, cloud-tenanted artifact published on no schedule. So the concrete regulatory ask is narrower and more checkable than attestation: just specify the registry. A maximum age, a refresh obligation, non-overlapping per-purpose allocation, dedicated rather than shared tenancy, and an append-only signed history so a range can't be retroactively disowned. Each of these is verifiable by a third party holding nothing but the published file, and none of them appear in any instrument I read.

5.1 Which clock actually binds

For a general-purpose AI model with systemic risk, the reporting duty is Article 55(1)(c) [11], which obliges the provider to keep track of, document, and report, without undue delay, to the AI Office and, as appropriate, to national competent authorities, relevant information about serious incidents and possible corrective measures to address them. Three details are easy to get wrong here: the duty runs to national competent authorities as well as the AI Office, it covers corrective measures and not only the incident itself, and it contains no fixed deadline. Article 73 is the separate regime for high-risk AI systems, with actual clocks of 15 days by default, 10 if a death is involved, and 2 for widespread infringements or serious disruption to critical infrastructure [12]. Article 91 is the Commission's power to request information. In an earlier draft of this report I cited Article 53 and a 24-hour deadline; Article 53 actually covers GPAI documentation and copyright obligations, and the 24 hours didn't exist. Both were wrong and I corrected them here.

So the operative deadlines for a GPAI provider are actually not in the Act itself. They are in the GPAI Code of Practice, which OpenAI signed in full and which sets five days for a cybersecurity breach and fifteen for serious harm to health, safety, rights, property or the environment [8]. That distinction matters for this kind of incident: a containment failure whose consequence is an intrusion into a third party's production infrastructure plausibly falls under the cybersecurity-breach branch and its five-day clock, not the fifteen-day harm branch. Which branch OpenAI's September 2026 filing was made under has not been stated publicly [8], and it is a question that can be answered. Its answer determines which clock applied to an incident where activity began 24 May and acknowledgement came 5 September. There is also a related question worth flagging rather than answering: serious incident is defined in Article 3(49) by reference to an AI system, while Article 55(1)(c) attaches to a GPAI model, and an escape during an internal evaluation with no deployment and no affected user sits awkwardly inside that definition. I'm not competent to resolve it, and it determines whether the duty attaches at all.

What my measurements do settle is narrower but clear: detection latency is not what makes any of these clocks hard to meet. A 17,600-event gap scan completes in 0.080 to 0.201 s across five runs, against a deadline measured in days, so the margin is roughly six orders of magnitude and no plausible hardware closes it. On the public record, leadership knew weeks before the company said anything [8]. Tooling is not the constraint; disclosure policy is.

5.2 Would a lab adopt it?

Publishing egress ranges is cheap and already common practice for crawlers, that's why real files exist to audit in the first place. Maintaining them to a specified freshness is a modest operational cost. Signing every action is not cheap, and a lab has a clear incentive not to do it under a regime where unsigned traffic from its own ranges becomes evidence against it. That tension is real and I don't resolve it. The registry requirements are the part a lab would plausibly accept, and they also happen to do most of the work.

5.3 Limitations

- 1 Evasion by egress. An agent reaching a platform from unobligated address space produces no gap: 1,000 actions, 0 gaps in one test, and 500 events from TEST-NET-2 against the real registry likewise 0. One commodity proxy defeats the method. This is the biggest limitation.
- 2 No assertable findings on today's data. Against real published ranges every gap is just a triage lead, so the method can't be deployed as an evidentiary tool until registry quality changes.
- 3 Attribution granularity. A gap attributes to the operator of an address range, not to a model or a run, and 14.6% of the audited space can't even tell two products from the same operator apart.
- 4 A gap is not an escape. Misconfiguration, a crashed sidecar or a stale registry all produce gaps. The report object returns its own caveats as data so a consumer can't present a finding stripped of them.
- 5 No real-log validation. All event-stream evaluation is synthetic; only the registry audit uses real data.
- 6 Composition policy calibrated on synthetic data. The procedure is measured and it corrected a placeholder that misfired 21% of the time, but the corpus is generated, so the numbers don't transfer. This module is thematically close to the PAAC audit's D-01/D-02 decomposition findings but does not defend against or discharge them and must not be cited as doing so.
- 7 Custody is in-process throughout. Every figure here was produced by a key extractable from process memory. The boundary exists and the external path is exercised, but against a software stub; no hardware assurance is claimed or verified.
- 8 Uncalibrated confidence scale. The three-level degradation and the 90-day staleness cut-off are my own judgement calls exposed as parameters.
- 9 Single machine, single operator. Ed25519 throughput varies by platform, and the registry audit covers one operator's two files, so I can't say whether 315-day staleness is typical or an outlier.

5.4 Future Work

The main open item now is measuring false positives against real logs from one cooperating platform, since the public-log route is closed. After that, recalibrate the composition policy against that platform's real benign traffic instead of a synthetic corpus. I also want to bind the external signing backend to an actual PKCS#11 token so a hardware-backed claim is earned rather than just configured. And extend the registry audit across multiple operators to find out whether the staleness I measured here is typical, since a single-operator sample can't tell.

6. Conclusion

This project is built on my previous project, PAAC: a deterministic safety wrapper for self-modifying AI agents, which intercepts every proposed code modification, compiles it to the Safe Intermediate Language (SIL), and verifies it against safety axioms using Z3 SMT-based bounded model checking. I modified it and named this project Kairos. The core finding is simple: making agents sign their actions doesn't help to catch an agent that has already escaped its environment. I built the signature system, tested it against both 2026 incidents, and it returned zero attribution, completely blind. What actually works is the opposite approach: look for the case where an agent was supposed to sign everything coming from its IP range but didn't. Gap detection recovered the full 15,000-action campaign with zero false positives and dated it back to the very first edit.

But gap detection has one problem: it only works if the IP range files that labs publish are accurate and up to date. When I actually checked the real files OpenAI publishes, they were over 300 days old, some addresses were shared between multiple products, and most were sitting on cloud servers that could change hands at any time. So even though the detection method works perfectly, none of its findings could be officially filed today, because the underlying data is too unreliable. The fix isn't complicated. Regulators just need to tell labs: keep your IP range files fresh, don't share addresses between products, and make the history tamper-proof. Anyone can verify all of that from the published file alone.

Code and Data

Code: `src/attribution/` (seven modules) with tests in `tests/test_attribution_incidents.py`, `tests/test_attribution_performance.py`, `tests/test_range_registry.py` and `tests/test_custody_and_calibration.py`. Apache-2.0. 81 tests pass in 10 to 31 s depending on machine load; the suite prints its own environment banner and every performance figure it reports, so no number in Section 4 is transcribed by hand. Expect your absolute latencies to differ from Table 1; the ratio in Section 4.2 is what should reproduce.

Data: cached snapshots of the two real operator-published egress range files in data/registries/, each recording its source URL so the audit can be reproduced against live data. Figure 1 is generated by scripts/make_figure.py, which computes panel (c) from those files at run time.

Code repository: not publicly released for this submission. A public repo is optional for this sprint. Nothing in the artifact is an installation recipe or an exploit, so there is no disclosure concern in releasing it later; the bundled range files are published by their operator for exactly this kind of consumption.

References

- [1] Proof-Carrying Agent Actions: Model-Agnostic Runtime Governance for Heterogeneous Agent Systems. arXiv:2606.04104. <https://arxiv.org/pdf/2606.04104>
 - [2] CAVA: Canonical Action Verification and Attestation for Runtime Governance of Agentic AI Systems. arXiv:2607.13716. <https://arxiv.org/html/2607.13716>
 - [3] AEX: Non-Intrusive Multi-Hop Attestation and Provenance for LLM APIs. arXiv:2603.14283. <https://arxiv.org/html/2603.14283>
 - [4] Attesting Outputs and Delegation Ancestry in Multi-Agent AI Systems. arXiv:2608.30387. <https://arxiv.org/html/2608.30387v1>
 - [5] Agent Identity Protocol: Verifiable Delegation for AI Agent Systems. IETF draft-prakash-aip-00. <https://www.ietf.org/archive/id/draft-prakash-aip-00.html>
 - [6] Prove-Transform-Verify Protocol for Attested Agent Identity. IETF draft-anandakrishnan-rats-ptv-agent-identity-00. <https://www.ietf.org/archive/id/draft-anandakrishnan-rats-ptv-agent-identity-00.html>
 - [7] ARMO. AI Agent Escape Detection: How to Catch Agents Breaking Their Boundaries. <https://www.armosec.io/blog/ai-agent-escape-detection/>
 - [8] The Next Web (September 2026). OpenAI has filed an EU incident report on the hijacked German wiki, the Commission says. <https://thenextweb.com/news/openai-eu-incident-report-german-wiki> Source for the GPAI Code of Practice five-day and fifteen-day deadlines, for the filing branch not having been stated publicly, and for the ~18,000 figure. Corroborated on the Article 55 standard by TechTimes, 8 September 2026, <https://www.techtimes.com/articles/326933/20260908/openai-files-first-eu-ai-act-incident-report-chief-scientist-admits-monitoring-gap.htm>
 - [9] OpenAI GPTBot egress ranges. <https://openai.com/gptbot.json> Cached snapshot, creationTime 2025-10-30.
 - [10] OpenAI SearchBot egress ranges. <https://openai.com/searchbot.json> Cached snapshot, creationTime 2026-01-02.
 - [11] Regulation (EU) 2024/1689, Article 55: Obligations of providers of general-purpose AI models with systemic risk. Official text via the European Commission AI Act Service Desk. <https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-55> Article 55(1)(c) quoted verbatim in Section 5.1.
 - [12] CIO (September 2026). EU AI Act Article 73: the new AI incident reporting deadline. <https://www.cio.com/article/4218777/the-eu-ai-act-just-gave-you-a-breach-notification-clock-you-didnt-know-about.html>
 - [13] Hugging Face. Security incident, July 2026. <https://github.com/huggingface/blog/blob/main/security-incident-july-2026.md>
- Incident dates and the 17,600 count are taken from the sprint brief and [13]; the wiki volume discrepancy is discussed in Section 1.

Appendix A: Dual-Use Considerations

The limitations required by the submission checklist are in Section 5.3; this appendix covers dual use.

The gap primitive is a monitoring capability over traffic an operator has committed to attesting, so its abuse surface is mainly false accusation and surveillance creep. Section 4.4 makes the false-accusation risk concrete rather than theoretical: a 315-day-stale range in reassignable cloud space can produce an attributable-looking finding against whoever now holds that address. That is why `is_assertable` refuses low-confidence gaps by default, and why the registry must be signed and append-only before anyone files on its output. Requiring per-action attestation also creates a detailed behavioural record of agent activity, which is competitively sensitive and a subpoena target; hashing parameters rather than recording them, as implemented, limits but does not remove this. Timing analysis over platform logs is a general traffic-analysis capability with obvious application to non-agent users, and I would not publish a tuned detector without that being considered. Nothing here assists an intrusion: the artifact analyses logs a platform already holds, the bundled range files are published by their operator for this kind of consumption, and there is no exploit, no installation recipe, and no capability that helps an agent escape.

Appendix B: Reproducing

```
pip install -r requirements.txt
```

```
python -m pytest tests/test_attribution_incidents.py tests/test_attribution_performance.py tests/test_range_registry.py tests/test_custody_and_calibration.py -v -s
```

Expected: 81 passed, 10 to 31 s depending on load. The environment banner test prints the measurement environment; compare it against Section 3.4 before treating Section 4 figures as comparable. The registry audit clock is pinned to 2026-09-11 so the staleness figures reproduce exactly; to audit live data, refetch from the source URL recorded in each file under `data/registries/` and expect larger staleness numbers. Console output is ASCII by design, because emoji in print statements raises `UnicodeEncodeError` under the Windows `cp1252` console encoding, which is how an earlier version of this suite failed silently.

LLM Usage Statement

I wrote this report. This project is built on my old project PAAC, which I built myself, as I mentioned in the conclusion. I used Opus 5 in the Kiro IDE during this sprint for the code, for most of the tests, for finding sources, and for reviewing my drafts and pointing out errors and gaps in them. I used a grammar tool on my own text, and it did not change my sentences. The reason I leaned on the model for the code and tests is that in three days it was hard for me to read my old project repository fully, and I did not want to miss any details, numbers or benchmarks that I used. On verification: every figure in Section 4 is printed by the test suite on my machine rather than transcribed by hand, the 81 tests pass on my machine, the Article 55(1)(c) wording was checked against the Commission's official text, and the cached range files match what OpenAI publishes. The model also caught two of my own errors that are now corrected in the report: an incorrect citation of Article 53 with a 24-hour deadline that does not exist, and an off-by-one in the composition detector's evidence chain.

Research conducted at the AI Incident Response Sprint, September 2026.