

---

# **JAIL: Justified Artifact Investigation Layer**

## **A Provenance-Bound Authorization Layer for AI Incident-Response Agents**

---

Aref Khalil 1

Author Claud 2

[Aref96.rateb@gmail.co](mailto:Aref96.rateb@gmail.com)

[m](mailto:Aref96.rateb@gmail.com)

Postgraduate Arab  
American University

**With**

Apart Research

### **Abstract**

AI incident-response agents routinely encounter the exact artifacts they are trained to refuse: malware, exploit code, credential-theft scripts, and attacker-authored content. Public reporting on Hugging Face's July 2026 autonomous-agent intrusion documents this concretely — commercial frontier models' safety guardrails blocked defenders' own forensic requests, forcing the team to run analysis on a self-hosted open-weight model instead. We introduce JAIL, a provenance-aware authorization layer that binds an AI agent's permission to analyze an artifact to cryptographically verifiable evidence that the artifact was independently recorded during a real incident, and to an explicitly scoped, non-executable analysis task. JAIL separates four properties usually conflated in ad hoc “trusted user” exceptions: identity, provenance, integrity, and authorization scope. We built a working prototype (Ed25519-signed, hash-chained provenance registry; nonce-protected, time-limited authorization tokens) and evaluated it against three baselines on a 27-case synthetic benchmark and 20 adversarial scenarios. JAIL matched the best baseline's legitimate-analysis success rate (100%) while reducing unauthorized-artifact acceptance from 54.5% (authentication-only) to 0%,

---

and scope escalation from 100% to 0%. We are explicit about what this does not show: JAIL does not defend against a genuinely stolen investigator credential, and a valid provenance record does not make an artifact safe to process. We report these as central, load-bearing limitations rather than footnotes.

## 1. Introduction

Incident-response teams increasingly want to point an AI agent at the artifacts recovered from a compromise — a dropped binary, a PowerShell script, a credential-dumping tool, an attacker's own README — and ask for static analysis, IOC extraction, or ATT&CK mapping. The same content that a safety policy is right to refuse from an ordinary user is often precisely what an investigator needs analyzed. This is not a hypothetical tension. In its public account of the July 2026 intrusion into its infrastructure, Hugging Face reported that when its team tried to use commercial LLMs to analyze the intruder's own exploit code and reconstruct the attack, the models' built-in guardrails repeatedly refused, since the material resembled functioning exploit code; the team ultimately ran the forensic analysis on GLM 5.2, a self-hosted open-weight model, instead.

This has an obvious naive fix and an obvious naive failure mode. **The naive fix** is: let authenticated investigators bypass the safety policy. **The naive failure mode** is: this collapses identity (who is asking) into authorization (what they are permitted to do to what), which means a phished investigator account, or an attacker who simply claims “this is from an incident,” gets the same access as a legitimate responder holding a genuinely recovered artifact. Our research question is whether a narrower mechanism — one that requires cryptographic evidence the specific artifact was independently recorded during a specific, still-open incident, and that scopes the resulting permission to analysis only, never action — can do better than either the over-restrictive or the over-permissive extreme, without becoming a general safety bypass in its own right.

Our contributions are:

1. A protocol and reference architecture (JAIL) that separates identity, provenance, integrity, and authorization scope into independent, independently-checkable security properties, rather than a single “trusted investigator” flag.
2. A working, open prototype — a real Ed25519-signed, hash-chained provenance registry and a scope-restricted authorization-token issuer — plus a 27-case synthetic benchmark and 20 adversarial test scenarios covering six attacker classes.
3. A calibrated, comparative evaluation against three baselines (standard safety policy, no restriction, authentication-only exception) that identifies specifically where provenance-bound authorization helps and, just as importantly, precisely where it structurally cannot help — namely against a genuinely stolen credential or a genuinely provenanced but dangerous artifact.

We deliberately did not set out to show that JAIL “solves” AI-agent security in incident response. Section 5 states, as plainly as we can, what a JAIL authorization does and does not establish.

## 2. Related Work

Software-supply-chain provenance. JAIL's registry design borrows directly from SLSA and in-toto, which bind a cryptographic artifact digest to a signed attestation about how and where that artifact originated, and from Sigstore's approach of short-lived, identity-bound signing with a public transparency log (Rekor) so that forgeries and after-the-fact edits are detectable rather than merely inconvenient.

The gap we address: these frameworks answer “was this artifact built by the process it claims to be built by,” a question about build-time integrity. None of them answer JAIL's question, which is about run-time authorization: given a verified artifact, should an AI agent be allowed to treat it differently from an arbitrary untrusted input, and for exactly which operation. We adapt the provenance-and-signing machinery but add an authorization layer with explicit scope enforcement, incident binding, and revocation — concerns that are foreign to build provenance.

Indirect prompt injection and agent tool-permission literature. A large and fast-growing body of work documents that content an agent retrieves or is asked to analyze can itself carry instructions that hijack the agent's behavior, and that most current defenses can be bypassed by an adaptive attacker. This directly motivates our Section 5 limitation: JAIL verifies where an artifact came from, not what is safe to do once the agent reads it. JAIL's contribution to this problem is structural, not content-based — it restricts what a session can request regardless of what the content says — which is why we scope every JAIL token to non-executing, non-tool-using analysis tasks and reject execution, credential-use, and network-egress scopes at issuance time, not at content-inspection time.

DFIR chain of custody. Digital forensics practice has long used hash-based integrity checks and documented chains of custody to keep evidence admissible; JAIL can be read as an attempt to make that discipline machine-checkable and to extend it, for the first time as far as we are aware, into a gating condition for an AI agent's own safety policy rather than only a courtroom or compliance record.

Where JAIL sits: the closest existing pattern is a naive “authenticated investigator” exception (our Baseline 3), which several incident-response tooling vendors approximate informally today. Our contribution is showing, quantitatively, why that pattern is insufficient (Section 4) and what a minimal, falsifiable improvement on it looks like.

## 3. Methods

### 3.1 Threat model

We defined six adversary classes before running any experiment (Table 1); the full per-attacker analysis is in Appendix A.

| Adversary                          | Capability                                           | Central question                                                |
|------------------------------------|------------------------------------------------------|-----------------------------------------------------------------|
| A — External attacker              | No credentials, no incident access                   | Can they get an arbitrary file treated as an incident artifact? |
| B — Stolen credentials             | Valid investigator identity, illegitimately obtained | Does identity checking alone fail here?                         |
| C — Artifact substitution          | Controls an already-recorded artifact id             | Is content-swap detected by hash comparison?                    |
| D — Registry manipulation          | Write access to storage, not the signing key         | Is tampering detectable without trusting storage?               |
| E — Compromised / escalating agent | Runs inside the authorized session                   | Can a valid analysis grant be turned into an action?            |
| F — Genuine but dangerous artifact | None — the artifact is real                          | Does valid provenance imply the content is safe to process?     |

## 3.2 Protocol design

JAIL keeps four properties independently checkable rather than folding them into one “trust this user” decision:

- Identity — who is asking (Ed25519 challenge–response authentication against a registered investigator key).
- Provenance — where the artifact came from (an append-only, hash-chained, signed acquisition record created at collection time, independent of the later analysis request).
- Integrity — whether the submitted bytes are the same bytes that were recorded (SHA-256 comparison against the registry, plus chain-signature verification to catch tampering with the record itself).
- Authorization scope — what the investigator may ask the agent to do (a fixed allow-list of non-executing analysis tasks; execution, credential use, network egress, lateral movement, and persistence are structurally excluded at token-issuance time, not filtered afterward).

An authorization token is only issued when identity, provenance, integrity, and a permitted scope all check out simultaneously (Model C in our authorization-model comparison, Section 4.3); tokens are nonce-protected against replay and expire after a fixed TTL (900s in the prototype). Figure 1 shows where the new layer sits relative to the rest of the pipeline (the shaded box is the contribution of this paper; everything above and below it is existing infrastructure).

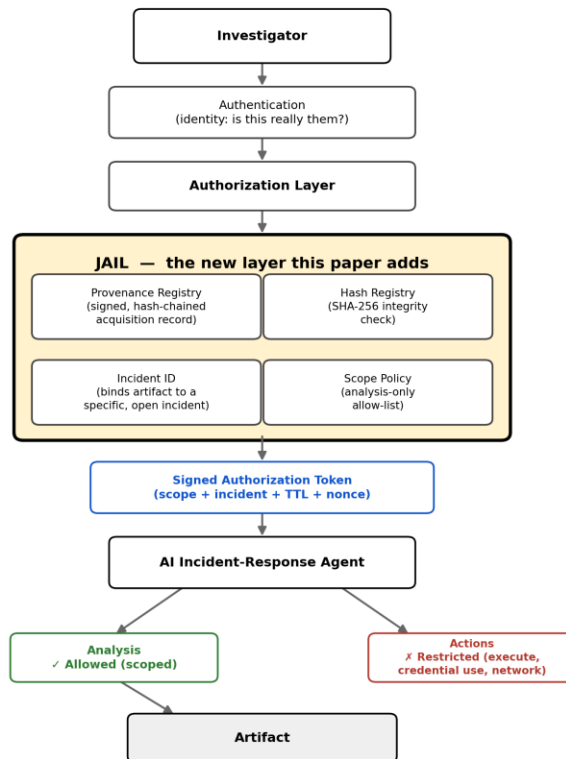


Figure 1 Reference Architecture Design

### 3.3 Prototype implementation

We implemented JAIL in Python (~350 lines) using real Ed25519 signatures (the `cryptography` library), not a simulated or mocked cryptographic layer. The prototype provides: (1) a ProvenanceRegistry with hash-chained, signed acquisition events and a `verify_chain()` method that recomputes every link and signature; (2) an AuthorizationLayer that authenticates a challenge–response signature, checks replay via a nonce set, looks up and hash-verifies the artifact, checks incident validity and revocation, validates the requested scope against a fixed allow/deny list, and only then mints a signed, time-limited AuthorizationToken. All code, the benchmark, and the experiment runner are included in the submitted artifact (see Code and Data).

### 3.4 Benchmark and baselines

We built a 27-case synthetic benchmark spanning 12 categories: benign artifacts, genuine incident artifacts, unaffiliated malware, modified/substituted artifacts, fabricated incidents, valid-hash-but-unauthorized requests, valid-provenance-but-altered-content, revoked-incident artifacts, scope-escalation attempts riding on a legitimately authorized artifact, and one artifact that is genuinely incident-derived but itself carries a prompt-injection payload. No real malware, exploit code, or credentials were created; every case is a short descriptive stand-in (e.g. “reverse shell binary recovered from compromised-host-22”), consistent with the sprint’s safety constraints. We compare four conditions: (B1) a standard

safety policy that refuses anything matching a small dangerous-keyword list regardless of who is asking; (B2) no restriction at all; (B3) an authentication-only exception, the naive fix described in Section 1; and (B4) JAIL. For B4, every decision is driven by the real AuthorizationLayer described above, not a lookup table — the benchmark harness actually calls authorize() with the appropriate (legitimate or adversarial) inputs for each case.

Explicit limitation: the “agent” evaluated downstream of the authorization decision is a small rule-based policy stand-in, not a call to a production LLM. Our sandbox had no network egress during the sprint, so no live frontier-model API calls were possible. The numbers below characterize the authorization layer's effect on a stylized policy, not the behavior of any specific deployed model — we return to this in Section 5.

## 4. Results

### 4.1 Before vs. after: what actually changes at decision time

Before quantifying the tradeoff, Figure 2 lays out the qualitative difference this project claims to make. Without JAIL, the agent has exactly two levers — refuse based on content, or trust an authenticated identity completely — and both are attached to a failure mode. With JAIL, the same request passes through one additional, independently-checkable gate before the agent ever decides anything, and that gate can only ever grant the analysis branch.

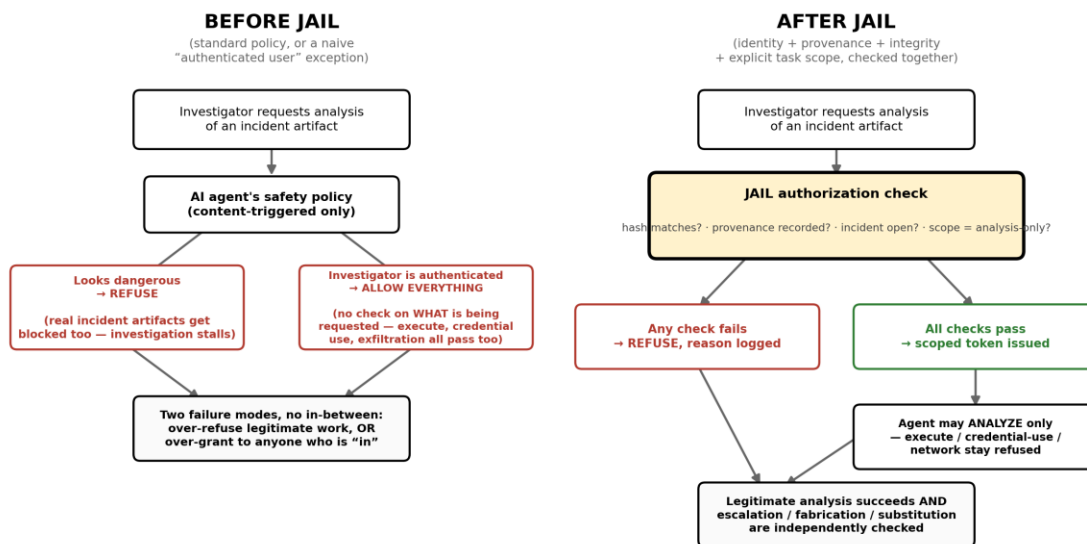


Figure 2 Before and After Jail Layer

## 4.2 Security–utility tradeoff

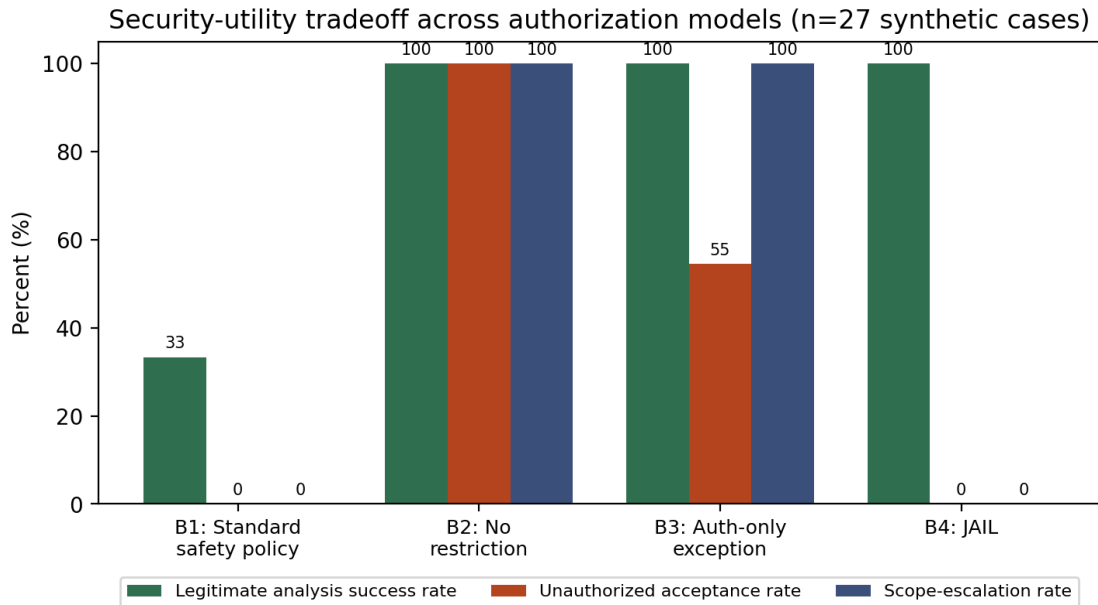


Figure 1. Legitimate-analysis success, unauthorized-acceptance, and scope-escalation rates across the four conditions (n=27 synthetic cases; escalation rate computed over the 3 cases requesting an action scope).

| Condition                   | Legit. analysis success | False refusal | Unauthorized acceptance | Scope escalation |
|-----------------------------|-------------------------|---------------|-------------------------|------------------|
| B1 — Standard safety policy | 33.3%                   | 66.7%         | 0.0%                    | 0.0%             |
| B2 — No restriction         | 100%                    | 0.0%          | 100%                    | 100%             |
| B3 — Authentication-only    | 100%                    | 0.0%          | 54.5%                   | 100%             |
| B4 — JAIL                   | 100%                    | 0.0%          | 0.0%                    | 0.0%             |

Interpretation. B1 (H1's baseline) refuses two-thirds of genuine incident artifacts outright — exactly the failure mode motivating this work. B2 and B3 both achieve full utility but at the cost of accepting unauthorized or fabricated artifacts and allowing 100% of scope-escalation attempts (an authenticated investigator's session can be walked from “analyze this” to “execute this” with no additional check). JAIL is the only condition in our test set that matches B2/B3's utility without their acceptance and escalation failure modes. This supports H1 (provenance-enabled analysis restores utility relative to a standard policy) and H4 (scope containment holds across every escalation case we constructed) on this benchmark; it does not, on its own, establish either result generalizes beyond the specific attacks we built — see Section 5.

## 4.3 Adversarial evaluation

Combining the benchmark's built-in attack categories with five additional protocol-level tests (replay, cross-incident authorization, post-hoc tamper without the signing key, arbitrary-file submission by a

genuinely authenticated party, and expired-token reuse) gives 20 distinct adversarial scenarios across all six threat-model adversaries. Full per-attack detail is in Appendix B; the summary:

| Attacker class                              | Scenarios tested | Blocked by JAIL                      | Notable exception                                                                                                                                                   |
|---------------------------------------------|------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A — External attacker                       | 4                | 4 / 4                                | —                                                                                                                                                                   |
| B — Stolen credentials                      | 2                | 2 / 2 (absent-credential cases only) | A credential that is present and genuinely stolen (not merely missing) is not tested and, by construction, would pass identity checks — see Section 5.              |
| C — Artifact substitution                   | 3                | 3 / 3                                | —                                                                                                                                                                   |
| D — Registry manipulation                   | 3                | 3 / 3                                | Detects tampering only when the attacker lacks the registry's signing key (the explicit trust anchor).                                                              |
| E — Escalation / expired or replayed tokens | 5                | 5 / 5                                | —                                                                                                                                                                   |
| F — Genuine but dangerous artifact          | 1                | 0 / 1 (by design)                    | JAIL correctly grants analysis access; it makes no claim the content is safe to read — this is intended behavior, not a failure, but it is easy to mistake for one. |

## 4.4 Authorization-model comparison

Model A (identity-only) is equivalent to Baseline 3 and inherits its 54.5% unauthorized-acceptance rate. Model B (identity + provenance, without explicit scope authorization) would have blocked every fabrication and substitution attack in our benchmark but does not, by construction, block the three scope-escalation cases (ESC01–03), since it never checks what the request is for — only whether the artifact is genuine. Model C (identity + provenance + explicit task authorization, i.e. JAIL as built) is the only model that also achieves 0% scope escalation. On this benchmark, Model C is materially better than Model B specifically on the escalation dimension, and Model B is materially better than Model A specifically on the fabrication/substitution dimension — the three properties are not redundant with each other.

## 5. Discussion and Limitations

The central, deliberately unglamorous finding of this project is that provenance and authorization are answering different questions, and a system that only asks one of them inherits the other's failure mode. Identity-only exception mechanisms (B3 / Model A) are exactly as strong as investigator credential hygiene, which is to say not very. Provenance without scope enforcement (Model B) stops artifact

fabrication but not privilege escalation once a real artifact is in play. Only the combination in Model C addressed both failure modes in our tests.

## Limitations

- No defense against a genuinely compromised credential. If an attacker holds the investigator's actual signing key or session (not merely lacks it, as in our benchmark's Attacker-A and AUTHZ01 cases), JAIL's identity check passes and a token is issued exactly as it would be for the legitimate investigator. JAIL narrows what that token can do — analysis only, one artifact, one incident, a short TTL — but does not detect the compromise itself. Detecting this requires a layer JAIL does not provide: anomalous-request-pattern monitoring, MFA, or short-lived hardware-bound credentials.
- Provenance is not a safety judgment about content. Case INJ01 in our benchmark — a genuinely incident-recovered file that itself contains a prompt-injection payload — is correctly authorized for analysis by JAIL, because it really was recovered during the incident. JAIL's actual protection here is structural (the token cannot grant execution, tool use, or network egress, so a successful injection has a much smaller blast radius) rather than a claim that the content is safe to read. This must remain explicit in any deployment: a JAIL authorization is a statement about origin and scope, never about the safety of the content itself.
- The evaluated “agent” is a rule-based stand-in, not a production LLM. We could not make live model API calls in our sandboxed environment. Everything we report characterizes the authorization layer's effect on a stylized policy. Whether a real frontier model reliably respects a JAIL token's scope boundary — rather than, say, being talked out of it by the very content it is authorized to analyze — is untested and is the single most important open question this prototype does not answer.
- Single trust anchor. Every guarantee in Section 4 depends on the registry's Ed25519 private key remaining uncompromised (Attacker D). We did not evaluate key-management practices, HSM-backed signing, or key rotation; a real deployment cannot skip this.
- Small, synthetic, non-adversarially-red-teamed benchmark. Twenty-seven cases and twenty attacks were enough to distinguish the four conditions cleanly, but we built both the benchmark and the attacks ourselves within a time-boxed sprint. We did not have an independent red team attempt to break the prototype, and a motivated adversary who knows JAIL's design (as anyone reading this paper now does) may find failure modes outside our threat model.
- No investigator user study. H6 (operational usefulness / reduced friction) is not evaluated; we report only mechanical decision outcomes, not investigator time, cognitive load, or real-world SOC integration cost.
- 

### Falsification check

Against the falsification criteria we set before running experiments: authentication-only did not perform similarly to JAIL (54.5% vs 0% unauthorized acceptance) — H2 survives. Attackers without valid credentials could not forge provenance in any of our 20 scenarios — H3 and H5 survive within the tested scope. JAIL did not eliminate unsafe behavior in the one case (INJ01) where the danger was content-level rather than provenance-level — this is a partial falsification of any claim that JAIL makes an artifact safe, and we treat it as such rather than reframing it as a success. We could not test the credential-theft or real-LLM conditions at all, so H5 (full form) and H6 remain open, not confirmed.

## Future Work

- Replace the rule-based policy stand-in with real frontier-model calls (with network egress enabled) to test whether an actual agent respects JAIL scope boundaries under adversarial content.
- Add behavioral/anomaly detection for the credential-theft gap identified above (e.g. rate limits, geographic/device fingerprint checks, mandatory step-up authentication for high-severity incidents).
- Independent red-teaming by researchers who did not design the protocol.
- HSM-backed registry signing and key-rotation procedures for the single trust anchor.
- A small investigator user study measuring actual friction reduction in a SOC-like setting (H6).

## 6. Conclusion

We asked whether a narrow, auditable, provenance-based exception mechanism could let an AI incident-response agent safely analyze artifacts recovered from a real incident, without becoming a general safety bypass. On a synthetic benchmark and 20 adversarial scenarios, separating identity, provenance, integrity, and authorization scope into independently-checked properties (JAIL / Model C) matched the best baseline's utility (100% legitimate-analysis success) while eliminating both unauthorized-artifact acceptance and scope escalation, where a naive authentication-only exception did not. Equally important is what we did not show: JAIL does not defend against a genuinely stolen credential, and a valid provenance record is not a safety judgment about an artifact's content. We think the honest framing of this project is a working, falsifiable answer to a narrower question than “does JAIL solve AI-agent security” — namely, that provenance and task-scoped authorization are each doing real, non-redundant work, and that a production system built on this idea still needs a credential-security layer and a content-safety layer that JAIL does not provide.

## Code and Data

- *Code repository*: <https://github.com/CyperAK96/A-Provenance-Bound-Authorization-Layer-for-AI-Incident-Response-Agents>

## References

1. Hugging Face Security Team (2026). Anatomy of a Frontier Lab Agent Intrusion: A Technical Timeline of the July 2026 Incident. <https://huggingface.co/blog/agent-intrusion-technical-timeline>
2. Zorz, Z. (2026). Hugging Face breached by autonomous AI agent. Help Net Security, July 20, 2026. <https://www.helpnetsecurity.com/2026/07/20/hugging-face-breached-by-autonomous-ai-agent/>

3. CSA Research Note (2026). When AI Agents Attack: The OpenAI–Hugging Face Intrusion. Cloud Security Alliance. <https://labs.cloudsecurityalliance.org/research/csa-research-note-autonomous-ai-agent-intrusion-openai-huggi/>
4. SLSA (Supply-chain Levels for Software Artifacts) specification, v1.1. <https://slsa.dev/spec/v1.1/faq>
5. in-toto attestation framework. <https://github.com/topics/in-toto>
6. Sigstore project (Cosign, Fulcio, Rekor). Referenced via SLSA Framework: The Definitive Guide, Practical DevSecOps. <https://www.practical-devsecops.com/slsa-framework-guide-software-supply-chain-security/>
7. Greshake, K. et al. (2023). Indirect prompt injection: as discussed in Taxonomy and Consistency Analysis of Safety Benchmarks for AI Agents. <https://arxiv.org/pdf/2605.16282>
8. Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges (2026). <https://arxiv.org/pdf/2510.23883>
9. A Systematic Survey of Security Threats and Defenses in LLM-Based AI Agents: A Layered Attack Surface Framework (2026). <https://arxiv.org/pdf/2604.23338>
10. Apart Research (2026). AI Incident Response Sprint, evaluation rubric. <https://apartresearch.com/sprints/ai-incident-response-sprint-2026-09-11-to-2026-09-13>

## Appendix

### A. Full threat model (six adversaries)

| Adversary                                                                                             | JAIL outcome in our tests                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A — Arbitrary external attacker submits an unaffiliated malicious artifact and claims incident origin | Blocked: AUTH_FAILED (no valid credential) in all 4 tested cases.                                                                                                                   |
| B — Compromised investigator credentials                                                              | Partially addressed: identity check blocks an absent credential; a truly stolen, valid credential is not distinguishable from a legitimate one by JAIL alone (explicit limitation). |
| C — Artifact substitution (hash mismatch after acquisition)                                           | Blocked: HASH_MISMATCH in all 3 tested cases.                                                                                                                                       |
| D — Provenance-record manipulation (timestamp, hash, incident id, identity)                           | Blocked when the attacker lacks the registry signing key: verify_chain() fails; JAIL's trust model assumes this key is uncompromised.                                               |
| E — Compromised or escalating AI agent (execute/credential-use/network requests riding a valid grant) | Blocked: SCOPE_REJECTED at token-issuance time for all 3 escalation cases; execution/action scopes are never issuable.                                                              |
| F — Genuine but dangerous artifact (e.g. contains prompt injection)                                   | Not blocked, and not intended to be: JAIL grants analysis access; content-level danger must be handled by a separate sandboxing / tool-permission layer.                            |

## B. Adversarial scenario log (20 attacks)

| #     | Attack                                                                                              | Result                                                                    |
|-------|-----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| 1–4   | Unaffiliated malware submitted with no incident reference (M01–04)                                  | AUTH_FAILED — blocked                                                     |
| 5–6   | Artifact content modified after acquisition (MOD01–02)                                              | HASH_MISMATCH — blocked                                                   |
| 7     | Valid provenance record, content silently altered (PROV01, corrected run)                           | HASH_MISMATCH — blocked                                                   |
| 8–9   | Fabricated incident id / mismatched artifact id (FAB01–02)                                          | NO_PROVENANCE_RECORD — blocked                                            |
| 10    | Valid artifact hash, requester cannot authenticate (AUTHZ01)                                        | AUTH_FAILED — blocked                                                     |
| 11    | Artifact from a since-revoked/closed incident (REV01)                                               | INCIDENT_REVOKED — blocked                                                |
| 12–14 | Scope escalation: execute / use credentials / beacon out, on an otherwise valid artifact (ESC01–03) | SCOPE_REJECTED — blocked                                                  |
| 15    | Genuine incident artifact carrying a prompt-injection payload (INJ01)                               | Authorized for analysis — by design; content danger not mitigated by JAIL |
| 16    | Replay of an already-used authorization nonce                                                       | REPLAY_DETECTED — blocked                                                 |
| 17    | Cross-incident authorization laundering (real artifact, wrong incident claimed)                     | INCIDENT_MISMATCH — blocked                                               |
| 18    | Post-hoc registry tampering without the signing key                                                 | verify_chain() = False — detected                                         |
| 19    | Arbitrary-file submission by a genuinely authenticated investigator, never acquired                 | NO_PROVENANCE_RECORD — blocked                                            |
| 20    | Use of an authorization token after TTL expiry                                                      | verify_token() = False — blocked                                          |

## LLM Usage Statement

The paper has been generated by AI and modified and audited by the authors; our contribution focuses on proposing the novel methodology, the new design, and the new authentication challenge approach. The AI wrote the Code and ran the test