Execution Assurance: An Evidence Primitive for AI Agents

Intentrax Whitepaper | Execution Assurance
Published 2026-07-09 - Intentrax Labs - www.intentrax.com/labs

Why the record of what an autonomous agent actually did should be an engine-generated, independently verifiable artifact - not a log the agent writes about itself.

Executive summary

As organizations delegate consequential actions to autonomous AI agents, a gap opens between what an agent claims it did and what can be independently established. Application logs, transcripts, and orchestration traces are all written by the executing system, so they inherit its trust assumptions: to believe the record, you must first trust the party being examined. This paper argues that autonomous execution needs a distinct evidence primitive - a tamper-evident, replay-verifiable artifact produced by an engine separate from the executor, checkable by anyone without access to the runtime that created it. Intentrax is the reference implementation of this primitive under an open verification protocol (AEVN). The artifact it produces is the Autonomous Execution Proof (AEP). This document describes the problem, the properties an execution-assurance primitive must have, and the honest boundaries of what such a primitive does and does not establish. It is written for auditors, insurers, and enterprise buyers evaluating how AI-agent activity can be evidenced. Intentrax is at a pre-revenue, production-beta stage; nothing here should be read as a claim of scale, customers, or regulatory endorsement.

The problem: self-attested execution does not survive scrutiny

The prevailing way to know what an AI agent did is to read what the agent, or its host application, recorded. That record is convenient and useful for debugging, but it is structurally self-attested. The same system that took the action also authored the account of the action, and can in principle edit, omit, or reorder it. For an internal engineering audience this is acceptable. For a downstream party - an auditor reconstructing events, an insurer adjudicating a claim, a counterparty resolving a dispute - it is not, because the account and the actor are the same entity. Monitoring and observability tools do not close this gap; they make self-attested telemetry richer and more searchable, but they remain descriptions the runtime emits about itself. What is missing is not more logging. It is a record whose integrity does not depend on trusting the executor, and whose correctness can be re-established from the outside.

What an execution-assurance primitive must provide

Five properties distinguish an assurance primitive from a log. Independence: the artifact is generated by an assurance engine, not editable by the executing agent, so the actor is not the author of its own evidence. Cryptographic tamper-evidence: the relevant execution facts are canonicalized into a stable byte representation and hashed (SHA-256) to a proof hash, then signed, so any later alteration is detectable. Replayability: a verifier re-derives the governing decision from the recorded inputs rather than reading a stored conclusion, so the outcome is reconstructed, not merely asserted. Policy-bound authority: the proof records the exact policy version and the identity, delegation, and constraints under which the action was permitted. Portable evidence: the artifact is expressible in machine and human-readable forms (JSON, an AEP export format, PDF) that travel to audits, disputes, and claims. Together these turn 'the agent says it did X' into 'an independent engine produced a signed artifact from which anyone can re-derive that X occurred under policy version P, bound to principal A, and confirm the artifact is unaltered.'

The mechanism, in outline

Internally, execution is captured as an ordered trace. That trace is not the external product. It is normalized into an external proof: the Autonomous Execution Proof. The engine is deterministic by construction - no randomness, no dependence on wall-clock time or environment, no hidden external I/O in the decision path - so the same execution inputs yield the same proof hash on any conforming implementation. The governing decision follows a fail-closed model: an action is permitted only if both the execution evidence and the bound authority permit it; otherwise it is denied. This is a deliberate design bias toward more denials rather than false permits. Proofs are recorded in an append-only registry with checkpoints, so the historical record grows but does not silently change. A separate, developer-facing Action Receipt may summarize an event for ergonomics, but it is explicitly non-authoritative and never replaces the canonical AEP.

Independent verification

The defining test of an assurance primitive is whether a party who does not trust the executor can still confirm the record. Intentrax ships a standalone verifier that operates on repository-local artifacts using only hash recomputation. It requires no access to the origin runtime, no network connectivity, and no private key material. It recomputes hashes, re-derives the decision from recorded inputs, and fails closed on authority drift or any inconsistency in the evidence bundle. This 'fail-closed on drift' behavior is central: the verifier's default answer to ambiguous or tampered evidence is to withhold a pass, not to grant one. Verification is thus reproducible by an outside party rather than dependent on the vendor's own systems being online and honest at the moment of the check.

Boundaries and current stage

Honesty about limits is part of the primitive's value. An AEP establishes what was executed, under which policy version, bound to which authority, and that the artifact is unaltered and replayable. It does not certify that the underlying business decision was wise, that an outcome was correct in the world, or that any regulation was satisfied. It is not compliance certification, and this product makes no claim to guarantee compliance, eliminate risk, or

provide total security. External evidence ingested as an execution passport is bound to a proof but never replaces it. Intentrax is pre-revenue and in production-beta. The engine has recently begun signing real proofs; the offline verifier is operational. Assurance Levels and a broader trust network are planned and are labeled as such. Live production traffic and mutation are currently blocked pending controlled unlock. AEVN is an open protocol; Intentrax is its reference implementation, not its owner. Market forces such as insurer and auditor demand - not any regulatory mandate - motivate this work.

How to engage

For technical evaluators, the fastest path is the standalone verifier and the published conformance vectors: obtain an evidence bundle, run the offline verifier, and confirm that it recomputes hashes and re-derives the decision without touching a runtime. Developers can integrate an adapter with one command (npx @intentrax/sdk init <adapter>), with several one-command adapters available. For auditors and insurers, the portable AEP export (JSON, AEP-JSON, or PDF) is designed to be dropped into existing evidence workflows and checked independently. We welcome design-partner conversations at this stage and will describe the current state - including the blocks above - rather than a finished platform.