Governed Agent Action: Authority as Provenance

N71 Research

Technical Report TR-2026-02 June 2026

Abstract

TR-2026-01 establishes that every belief in the N71 graph carries a derivation chain ending in verbatim evidence. This report establishes the dual property for action: every side effect an agent produces carries an **authority chain** — a finite, inspectable derivation of *why it was allowed* — with the same rigor that evidence chains provide for *why it is believed*. The construction addresses the problem the agent industry has not formalized: delegation. Modern agent systems act through chains — a human grants standing authority, a synthesis process proposes work, a dispatcher assigns it, a coding agent executes it, a provider integration performs the final side effect — and at every hop, the question "on whose authority?" is conventionally answered by credential inheritance, which is to say, not answered. We specify a governance function computed per action over the full chain, five properties it satisfies (totality, instance-bound approval, scope monotonicity, delegation attenuation, attribution completeness), the two-plane execution architecture that enforces it, and adversarial walkthroughs showing how each failure class in our threat model is excluded by construction. The architecture is deployed in production under sovereign and air-gapped constraints in a regulated government environment. We publish the governance contract; policy computation internals are not disclosed.

1. The Delegation Problem

The economic case for organizational AI is work performed, not questions answered. The moment systems act, the industry's permission model collapses into a single anti-pattern: **credential inheritance**. An agent is connected with a person's credentials and thereafter acts *as* that person — for everything, indefinitely, at every depth of delegation. The CFO's assistant agent drafts an email: fine. The same agent, three hops later — a scheduled process it triggered, calling a sub-agent, calling an integration — wires a payment: same credentials, same "authority," no record distinguishing the hops.

The problem is not hypothetical and it is not solved by approval pop-ups. It is structural: in a delegated chain, *whose authority is the action exercising, and how much of it survived the journey?* Conventional answers fail in both directions. Inheritance over-grants — authority amplifies silently as it propagates. Hard-coded tool permissions under-specify — they answer "can this tool be called?" when the real question is "can this principal, through this chain, perform this operation on this resource, now?"

Our position: authority is a provenance problem, and it should be solved the way provenance is solved. TR-2026-01's first invariant states that every belief has a finite derivation path ending in source evidence. This report's central claim is the symmetric invariant:

I6 — Authority totality. Every side effect has a finite authority chain ending in an explicit human grant, and the effect's permissibility is computed from that chain — never inherited from a session, a credential, or a prior action.

2. Threat Model

The governance layer is specified against six failure classes we consider disqualifying. Each reappears in §5 as an adversarial walkthrough.

T1 — Unauthorized side effects. An agent performs a write outside the authority of the principal it serves. **T2 — Authority amplification.** Permission granted for one purpose silently generalizes — across operations, resources, time, or delegation hops. **T3 — Double execution.** The same unit of work is performed twice because claim semantics are weak. **T4 — Silent abandonment.** Work is claimed and dies, with nothing distinguishing "in progress" from "dead." **T5 — Unaccountable action.** An effect occurs that cannot be reconstructed — actor, authority, basis, result — from records. **T6 — Perimeter violation.** In sovereign deployments: data, telemetry, or model traffic crosses a boundary it must not cross.

3. The Governance Function

Every side-effectful operation is classified, before execution, by a function

> **G**(principal, agent, scope, operation, resource) $\rightarrow$ { **executed**, **approval-required**, **blocked** }

where *principal* is the human whose authority is being exercised, *agent* is the automated actor requesting (with its position in the delegation chain), *scope* is the workspace/resource/classification context, and *operation* is the verb class — read, draft, send, share, modify, delete are not one category. The outcomes: **executed** (performed, receipt recorded), **approval-required** (held as a structured pending object — full intended operation, requesting chain, motivating evidence — surfaced for human decision; the approval, with approver and basis, enters the graph beside the action), **blocked** (refused with a machine-readable reason; blocks are recorded, because a pattern of blocked attempts is itself organizational signal).

The contribution is not the three states. It is the five properties G satisfies, enforced by construction:

P1 — Totality with denial default. G is defined for every operation the system can express; an operation G cannot classify is blocked. There is no unclassified path and no permissive fallback.

P2 — Instance-bound approval. An approval attaches to one held action — this message, these recipients, this content hash. It does not generalize to the action's class, the session, or the agent. Re-performing an approved action is a new action requiring its own classification. (Excludes T2's most common form: the approval that quietly becomes a policy.)

P3 — Scope monotonicity. Narrowing scope never increases permission. An agent confined to a workspace cannot, through any composition of permitted operations, obtain an effect outside it. Reads obey the same law: results are bounded by authority at the source — filtered before they leave the system, not redacted after.

P4 — Delegation attenuation. Authority composes by intersection along the chain. When a principal grants an agent, which dispatches a worker, which calls an integration, the effective authority at the final hop is the *meet* of every grant along the path — never more than the weakest link, and every hop is recorded. Authority can only narrow as it propagates. This is the formal negation of credential inheritance, under which authority is copied whole at each hop.

P5 — Attribution completeness. Every classification — all three outcomes — produces a durable record binding the full chain: principal, every intermediate agent, scope, operation, evidence basis, decision, and (for executed actions) the receipt and produced artifacts. The record is written to the same evidence layer as everything else, which yields the property developed in §6.

The policy computation by which G maps inputs to outcomes is the proprietary core of the layer and is not specified. The properties above are its published contract: they are what an auditor can test and what we expect to be held to.

4. The Two-Plane Execution Architecture

G needs an execution substrate that cannot be routed around. N71 runs two planes; every side effect in the system traverses one of them, and both terminate in G.

The pull plane — MCP. Agents — frontier assistants, coding agents, open agents, customer-built agents — consume the graph through the primitive set of TR-2026-01 §3, with P3 enforced at retrieval and every write primitive returning a G outcome. An agent on this plane carries an explicit chain identity: it is *some principal's* agent, and its calls are classified as such.

The push plane — the dispatch daemon. Work the system itself proposes — Thoughts with actionable payloads, grounded in graph evidence — is executed by fleets of coding agents (Claude Code, Codex, and compatible open agents) under a supervised edge daemon. The work lifecycle is contractual: **proposed** (with its evidence chain), **classified** (G runs before any agent touches side-effectful work; approval-required work waits), **claimed** (lease-based atomic claim — exactly one worker holds a valid lease, atomic at the storage layer, with expiry; T3 is excluded by construction), **executing under heartbeat** (a worker that stops heartbeating forfeits its lease at expiry and the task returns to claimable — T4 excluded; the daemon itself runs under host-native process supervision, launchd/systemd, so daemon death means restart, not absence), staged **sandbox-first** where the work supports it, with promotion to live execution as a distinct, separately classified step, and **reported** — completion or failure closes through a verifying primitive that checks the claim, records outcome and artifacts, and writes the full P5 attribution. Failure reports must carry the blocker; a task that cannot explain its failure has not finished failing.

The daemon–server contract — claim semantics, lease and heartbeat obligations, report verification — is frozen and versioned, so heterogeneous worker fleets evolve independently against a fixed interface. That is what makes supervising agents you did not write feasible at all.

5. Adversarial Walkthroughs

A governance specification earns trust by showing its failures excluded, not asserted away. One walkthrough per threat class.

T1 — The over-eager draft-sender. A coding agent, completing a task scoped to *drafting* a customer update, attempts to send it. The operation class changes from draft to send; G classifies

against the chain's actual grant, which covers draft only. Outcome: approval-required. The held object carries the full message, the chain, and the motivating Thought; a human approves or doesn't. The audit record shows the attempt, the hold, and the decision — including if the decision was no.

T2 — The approval that tries to become a policy. A principal approves sending one renewal notice. The agent, pattern-matching, attempts the same operation for nine other accounts. P2: the approval was bound to the instance. Nine new classifications run; each lands at approval-required on its own. What inheritance-based systems record as "user enabled sending" is, here, ten records: one approval and nine holds.

T3 — The race. Two daemon workers observe the same proposed task within the same second and both attempt to claim. The claim is a single atomic storage operation; exactly one acquires the lease. The other's attempt fails cleanly and is logged. There is no window in which both believe they own the work, because ownership *is* the lease row, not a belief.

T4 — The dead worker. A worker claims a task and its host dies mid-execution. Heartbeats stop; at lease expiry the task returns to claimable, and the eventual completing worker's report supersedes nothing — the dead claim's history remains, attributed. The failure narrative is reconstructible: claimed, heartbeats ceased at t, lease forfeited, reclaimed, completed.

T5 — The Monday-morning audit. "What did our agents do last week, on whose approval, and what came of it?" Because P5 writes every classification into the evidence layer, this is an ordinary graph query, answerable with the same anchoring primitives as any other question — down to the verbatim approval and the produced artifacts. No log archaeology, no reconciliation across systems: the actions live where the knowledge lives.

T6 — The perimeter. In the air-gapped tier, an agent's task would benefit from an external lookup. There is no external path: inference, extraction, retrieval, and G itself all execute inside the perimeter — G is computed locally from explicit inputs and requires no external policy service. The operation either completes from interior resources or fails with a recorded reason. "No egress" is not a firewall rule layered onto a chatty architecture; it is the architecture, exercised.

6. The Self-Model Property

P5 has a consequence that is easy to miss and, we believe, is where this architecture points. Because every action re-enters Layer 1 as a raw event — receipt, artifacts, chain — the system's model of the organization *includes the results of the system's own actions*, subject to the same invariants (I1–I5) as everything else. The graph does not merely remember the company; it remembers what it did to the company, what was approved and refused, and what came of it. Every property TR-2026-01 establishes for organizational memory — provenance, bi-temporality, non-destructive history, validated emission — applies reflexively to the system's own behavior. An agent platform without this property can act; it cannot *learn from acting* with evidence-grade fidelity, and it cannot be audited as a single system of record. This reflexive loop is the foundation of our agent-improvement research direction, reported separately.

7. Sovereign Deployment

The architecture was not designed abstractly and hardened later; it was shaped under the constraint of a production deployment in a regulated government environment on sovereign cloud infrastructure. The pattern is two-tiered: a **connected sovereign tier** (jurisdictionally bounded

region, absolute data residency, in-boundary inference) and an **air-gapped tier** (no external network path: no egress of data, telemetry, or model traffic). Two properties make the air-gapped tier real rather than diagrammatic: the substrate reduces the model's task to reading comprehension over pre-structured context (TR-2026-01 §4), so the system operates accurately on the smaller models closed perimeters can host; and G computes locally (T6). We make a deliberately narrow claim: not "air-gap available," but that the governance and execution model specified in this report is the one running under those constraints, and is the reason the deployment exists.

8. Limitations and Ongoing Work

P1–P5 constrain G's behavior; they do not make its policy *right* — a misconfigured grant classified with perfect totality is still a misconfigured grant, and policy authoring remains a human responsibility the system can audit but not absolve. Outcome granularity is deliberately coarse; richer outcomes (execute-with-constraints, time-boxed grants) are under design, cautiously — each increment of expressiveness is an increment of audit complexity. Approval latency is a product problem as much as an architectural one: a governance layer that becomes a queue is a governance layer that gets bypassed. Cross-organization federation — governed slices of graphs shared across boundaries — is operational in early form and will be specified once the cross-boundary audit model is settled.

9. Disclosure

Published: the governance properties P1–P5, the invariant I6, the work lifecycle, the walkthrough behaviors, and the deployment constraints — everything an auditor would test. Proprietary: the policy computation, scope resolution internals, and the frozen daemon–server contract specification.

Companion reports: TR-2026-01 (the substrate and invariants I1–I5), TR-2026-03 (ontology induction under the same gate architecture).