Governed Multi-Agent Systems

A formal model of authority for software agents that act: capabilities, grants, bound approvals, separation of duties, delegation and provenance

Research paper LL-GMA-01
Version 1.0.0, 26 November 2026

Abstract

Software agents built on language models now propose acts with legal effect: they send documents for signature, file submissions, release funds and delete records. Whether such a system deserves trust depends not on the capability of its model but on whether every act can be traced to someone entitled to authorize it, under rules applied when it ran. This paper gives a formal model of authority for agent systems in which competence confers no right to act. Principals hold authority and agents hold none; a proposed act passes through capability modes composed by restriction, through tenant grants that deny by default and through approvals that are content-addressed, versioned, expiring acts of named humans.

We prove six results under stated assumptions. Restrictive composition never lets one rule widen what another forbids. An approval bound to the SHA-256 digest of an injective, deterministic encoding of the input cannot be reused for another input without a hash collision. No evaluator whose inputs omit the requester can enforce separation of duties without refusing legitimately approved acts; no requirement that leaves reviewers unnamed can require two distinct persons; delegation is safe and complete exactly when it intersects the child's policy with the parent's authority; and no authorization function that lacks a request's provenance can tell a principal's request from one induced by content the agent has read. Two of these, and the impossibility of enforcing attenuation without the parent's restrictions, follow from one lemma: an evaluator cannot enforce a norm that separates situations its inputs do not.

Lawie implements and tests the approval, grant and tenant controls described in section 10 [TESTED]. In the BEN-02 benchmark three of its authorization functions denied none of 96 legitimate requests, refused every scenario of some attack classes and allowed every scenario of others [MEASURED]. We close with ten requirements, each with its cost, and the risks that remain when all ten are met.

Prepared by Lawie Labs. *Status:* First edition. *Distribution:* Public. *Basis:* source repository `main` @ `ffa13d4bce` (2026-11-24); experiments of 26 November 2026; EUR-Lex texts retrieved 26 November 2026. *Audience:* Security engineers, researchers of AI agents, legal technologists, regulators and diligence teams.

Evidence labels. Every material statement about Lawie carries an evidence class. [IMPLEMENTED] in source at the basis commit; [TESTED] exercised by repository tests at that commit; [MEASURED] measured by an offline experiment of this suite on the stated public or synthetic data, never a production or customer measurement; [ANALYTICAL] follows by argument from stated assumptions; [SIMULATED] output of a stated model; [EXTERNAL] checked against a cited primary source; [PROPOSED] design or protocol not built or not run; [UNVALIDATED] a hypothesis, stated as such; [ABSENT] not implemented; [BLOCKED] awaits an external owner or decision. Statements describe the source repository at the basis commit, not a deployed service: no workflow described here has a dated production acceptance, nothing is independently audited, and no statement has been reviewed by an independent human expert. This is a technical publication, not legal advice, not an offer of services and not a statement that any use of Lawie complies with any law.

Position

> **Intelligence is not authority. Mission.** Make high-stakes institutional work executable by supervised machines: completed, approved, evidenced, defensible.

A language model can draft a contract, find a precedent and plan a filing. None of this gives it the right to send the contract, cite the precedent to a court or make the filing. Competence is a property of the model; authority is a relation between persons, institutions and rules, and it must be shown, not inferred. Five positions follow, each with its cost.

1. **Competence confers no authority.** A model's output is a proposal. It becomes an act only through the standing authority of a principal or the bound approval of a named human. *Cost:* consequential acts wait for human authority; throughput is limited by attention, not by model speed.
2. **Deny by default; compose by restriction.** Effective authority is the most restrictive applicable rule, and no applicable rule means refusal. *Cost:* work fails until someone grants it; a configuration that is too narrow shows itself as a refusal, while an over-broad grant does not show itself at all.
3. **An approval approves one act.** It names the tool, its version and the digest of the exact input, names its reviewer and expires. *Cost:* any change to the input needs a new approval.
4. **Authority is checked where it is used, against current state.** A revoked grant, a departed reviewer or a lapsed admission must stop the next call. *Cost:* every call pays for a lookup, and a failed lookup refuses.
5. **What the evaluator is not given, it cannot enforce.** Separation of duties needs the requester, attenuation needs the parent's restrictions, provenance-sensitive refusal needs the provenance. *Cost:* these facts must be recorded at their origin and carried through every hop between agents.

Contributions

K1 A formal model of authority for agent systems, with seven norms (section 2).

K2 An indistinguishability lemma from which the impossibility results for separation of duties and provenance, and the need to carry a parent's restrictions, follow (lemma 3.1).

K3 The binding result for approvals with its exact assumptions: collision resistance, an injective and deterministic encoding, recomputation from the input that executes (proposition 4.1).

K4 An expressiveness result for quorum (proposition 5.2) and a characterisation of safe delegation as intersection (proposition 7.1).

K5 A description of the controls one implementation provides, with its aggregate benchmark outcome (section 10).

Each result is elementary; we claim no new cryptography and no new access-control theory [1–5]. The contribution is to state, for systems in which a language model proposes the acts, what an authorization layer can provide given what it observes, and to test one implementation against it.

Where Lawie stands

Lawie implements most-restrictive composition [TESTED] and default-deny grants [TESTED]. Section 10 describes Lawie's controls and its aggregate BEN-02 outcome [MEASURED]. Nothing described here is in production [ABSENT] or independently audited [ABSENT].

Contents

Abstract **1**
Position **2**
1 Introduction **4**
1.1 Agents that act 4
1.2 Four assumptions that fail 4
1.3 Authority, not competence 4
1.4 Evidence and organisation 5
2 A model of authority for agent systems **5**
2.1 Principals, runs and acts 5
2.2 Approvals and approval requirements 5
2.3 Scenarios, views and norms 6
3 What an evaluator can enforce **6**
3.1 The indistinguishability lemma 6
3.2 Composition by restriction 7
4 Approvals as content-addressed acts **8**
4.1 The binding result 8
4.2 What the binding does not do 8
4.3 Lawie 9
5 Separation of duties and quorum **9**
5.1 Separation of duties needs the requester 9
5.2 Quorum needs counting 9
6 Tenant isolation **10**
7 Delegation and attenuation **11**
7.1 The attenuation result 11
8 Provenance of a request **11**
9 Risks specific to many agents **12**
10 Lawie's controls and its BEN-02 outcome **13**
10.1 What BEN-02 measured 13
11 Requirements and residual risks **13**
11.1 Ten requirements 13
11.2 Risks that remain when all ten are met 14
12 Limitations and conclusion **15**
References **16**

1 Introduction

1.1 Agents that act

An agent, here, is a program that turns the output of a language model into calls to tools. The calls that matter have effects outside the program: a contract sent for signature, a submission filed with a court or regulator, a payment released, a matter document deleted. A firm must later be able to defend each of these acts, and each is reversible only at a cost, if at all.

Two developments make the question of authority acute. First, the boundary between instruction and data has dissolved: an agent that reads a counterparty's e-mail, a retrieved judgment or a web page reads text that can contain instructions, and indirect prompt injection turns such text into a way of steering the agent's proposals [6]; even direct adversarial prompts redirect a model's goal [7]. Second, systems now compose agents: a planner delegates to specialists, which call tools and sometimes spawn further agents. Increasingly agentic systems open gaps in responsibility and erode accountability for the harms they cause [8]; excessive agency is listed among the principal risks of applications built on language models [9].

1.2 Four assumptions that fail

1. *A capable model can be told what not to do.* Instructions and data reach the model through one channel. A model that reads untrusted content can be induced to propose an act its principal did not want, so a governed system must treat every proposal as possibly adversarial and must not rest its safety on the model.
2. *A human confirmation is an approval.* A confirmation that does not bind the exact act authorizes whatever executes next; an approval that can be presented for another input is a signed blank cheque (section 4).
3. *Two approvals mean two people.* Only if the evaluator counts persons. One that checks two predicates against one pool of approvals is satisfied by one person approving twice (section 5).
4. *A sub-agent inherits its parent's safety.* Only if the parent's restrictions travel with the delegation and are intersected with the child's own. Otherwise delegation obtains authority the parent was denied: the confused deputy [3] in a new form (section 7).

1.3 Authority, not competence

The distinction formalised here is old. In legal theory a power to change legal relations exists because a rule confers it, not because its holder is able to exercise it [10], and being an authority on a question differs from having authority to act [11]. The protection literature distinguished the designation of an object from the authority to use it [3], showed in capabilities how to bind the two [1], and required that every access be checked against current authority, that the default be refusal and that critical operations need more than one key [2]. Commercial integrity policies made separation of duty central [4], and role-based access control expressed it and cardinality as constraints [5]. Code that implements such rules regulates what agents can do as surely as a statute regulates persons [12].

Lawie's stated mission is to make high-stakes institutional work executable by supervised machines. Its authority model binds every consequential act to a principal, a named human authority, a scoped and expiring approval receipt, the act and its artefact, and keeps approval, execution and finality distinct [PROPOSED]. That model is a design statement; section 10 describes the controls that implement it.

1.4 Evidence and organisation

Analytical results are propositions with proofs; they hold under their assumptions and say nothing about any deployed system. *Empirical* results come from the BEN-02 benchmark, run once offline on 26 November 2026 on 552 synthetic scenarios against components imported unchanged from the basis commit and reported in full in LL-BEN-02. *Implementation* statements describe source at the basis commit and carry the class assigned by the suite's claim register. Section 10 describes Lawie's controls. The correctness of what agents write is treated elsewhere in the suite (LL-REL-01, LL-BEN-01), the trust analysis in LL-SEC-01 and evidence chains in LL-EVP-01.

2 A model of authority for agent systems

The model describes one decision: whether a tool call that an agent proposes may execute. It abstracts from how the model produced the proposal, from concurrency and from the transport between components. Time enters as the instant t at which the decision is made.

2.1 Principals, runs and acts

Definition 2.1 (Principals and runs)**.** There is a finite set O of tenants (organisations), a finite set U of users (natural persons) and a membership relation that assigns each user roles in tenants. A *run* ρ is one execution of an agent on behalf of a principal. It has a tenant $o(\rho) \in O$, an acting user $u(\rho) \in U$ who is a member of $o(\rho)$, an agent definition, a run policy $\mathsf{Pol}(\rho)$ and, if another run spawned it, a parent $\pi(\rho)$; we write $\pi(\rho) \rightsquigarrow \rho$. Runs form a forest. An agent is not a principal: it holds no authority of its own.

Definition 2.2 (Acts)**.** There is a set $\mathcal{T}$ of tools, each with versions. An *act* is a triple $\alpha = (\tau, v, x)$ of a tool, a version and an input x, a JSON value. For each tool τ, X_τ is the set of its inputs up to the equivalence under which the tool behaves identically (for example, the order of an object's keys).

Definition 2.3 (Capability modes and their composition)**.** The modes $\mathsf{auto} < \mathsf{lim} < \mathsf{appr} < \mathsf{forb}$ (automatic, allowed within limits, requires approval, forbidden) are totally ordered. A *policy source* σ is a partial function that assigns a mode $\sigma(\alpha, \rho)$ where it applies: a tool's catalogue entry, the agent's definition, the run policy, a tenant grant, a data-boundary rule, a budget. For a finite set Σ of sources the *effective mode* is

$$m_\Sigma(\alpha, \rho) = \begin{cases} \mathsf{forb} & \text{if no } \sigma \in \Sigma \text{ applies to } (\alpha, \rho), \\ \max\{\sigma(\alpha, \rho) : \sigma \in \Sigma \text{ applies}\} & \text{otherwise.} \end{cases} \tag{1}$$

An act may execute only if $m_\Sigma \neq \mathsf{forb}$, and, when $m_\Sigma = \mathsf{appr}$, only with a valid binding approval.

Definition 2.4 (Grants)**.** A *grant* $\mathsf{grant}(o, \tau)$ records a state, a validity window $[t_e, t_x)$, a ceiling on the data classification and on the risk tier of acts it covers, and whether approval is required. At time t a grant admits α for a run of tenant o only if it exists, is active, $t_e \le t < t_x$, and the act is within both ceilings. A tool without an active grant is refused.

2.2 Approvals and approval requirements

Definition 2.5 (Approvals)**.** An *approval* is a record $a = (id, \tau, v, h, st, t_{\mathrm{req}}, t_{\mathrm{rev}}, t_{\mathrm{exp}}, \mathsf{rev})$: an identifier, a tool, a version, a digest h, a status $st \in \{\mathit{pending}, \mathit{approved}, \mathit{rejected}\}$, the instants at which it was requested, reviewed and expires, and the reviewer $\mathsf{rev}(a) \in U$. Let enc be an encoding of inputs into byte strings and $\mathscr{H}$ the SHA-256 function. Then

$$\mathsf{valid}(a, t) \iff st = \mathit{approved} \ \wedge\ \mathsf{rev}(a) \neq \bot \ \wedge\ t_{\mathrm{req}} \le t_{\mathrm{rev}} \le t < t_{\mathrm{exp}},$$
$$a \text{ binds } (\tau', v', x') \iff \tau' = \tau \ \wedge\ v' = v \ \wedge\ \mathscr{H}(\mathsf{enc}(x')) = h.$$

Definition 2.6 (Approval requirements)**.** Requirements are generated by $q ::= \mathsf{single}(ref) \mid \mathsf{anyOf}(q_1, \ldots, q_n) \mid \mathsf{allOf}(q_1, \ldots, q_n) \mid \mathsf{seq}(q_1, \ldots, q_n)$ with $n \geq 1$, where a reference *ref* may name an approval identifier, a tool and version, a digest and a reviewer. For a set G of approvals, an act α and a time t: $G \models \mathsf{single}(ref)$ if some $a \in G$ is valid at t, binds α and agrees with every field that *ref* names; anyOf holds if some child holds, allOf if every child holds, and seq if every child holds and the witnesses' review instants are non-decreasing in the order of the children. Witnesses of different children need not be distinct.

The last sentence is the natural semantics of a Boolean policy language, which is why the quorum problem of section 5 is easy to miss.

2.3 Scenarios, views and norms

Definition 2.7 (Scenarios, views, evaluators)**.** A *scenario* s is the complete situation of a decision: the act, the run and its lineage of ancestor runs with their policies, the approvals with all their fields, the set $R(s) \subseteq U$ of persons responsible for the request, the provenance $\mathsf{prov}(s) \in \{principal, content\}$ of the proposal, the time, and the state of grants and memberships. A *view* is a function view from scenarios to the inputs that an evaluator receives. An *evaluator* is a function E from views to {ALLOW, DENY}; the decision on s is $\mathsf{E}(\mathsf{view}(s))$.

Definition 2.8 (Norms and enforcement)**.** A *norm* is a predicate on scenarios; a scenario is *authorized* if every norm holds. An evaluator *enforces* a norm N on a set S of scenarios if $\mathsf{E}(\mathsf{view}(s)) = \mathsf{ALLOW}$ implies $N(s)$ for every $s \in S$, and is *non-obstructive* on S if it allows every authorized $s \in S$.

The norms are those of the BEN-02 protocol, stated here as predicates.

N1 Binding. Every approval relied on binds the act.

N2 Validity. Every approval relied on is valid at the time of the decision.

N3 Separation of duties. No approval relied on was reviewed by a person responsible for the request: $\mathsf{rev}(a) \notin R(s)$.

N4 Quorum. Where k approvals from an eligible set $E \subseteq U$ are required, the approvals relied on have at least k distinct reviewers, all in E.

N5 Tenant. The tenant requested and every tenant that the act designates, under any field name, equal $o(\rho)$.

N6 Delegation. A run holds no authority its parent does not hold (section 7).

N7 Provenance. An act induced by untrusted content rather than by the principal is refused or routed to a human approval.

The norms are Lawie Labs' statement of what authorization requires; the results below apply to any norm with the same structure. Figure 1 places the elements on the path from a principal to an executed act.

3 What an evaluator can enforce

3.1 The indistinguishability lemma

Lemma 3.1 (Indistinguishability)**.** *Let s and s' be scenarios with* $\mathsf{view}(s) = \mathsf{view}(s')$*. Every evaluator over* view *decides s and s' alike. Consequently, if s is authorized and s' violates a norm N, no evaluator over* view *is both non-obstructive on $\{s\}$ and enforces N on $\{s'\}$.*

Proof. E is a function of its argument, so $\mathsf{E}(\mathsf{view}(s)) = \mathsf{E}(\mathsf{view}(s'))$. Non-obstruction on s requires ALLOW; enforcement on s' requires DENY. □

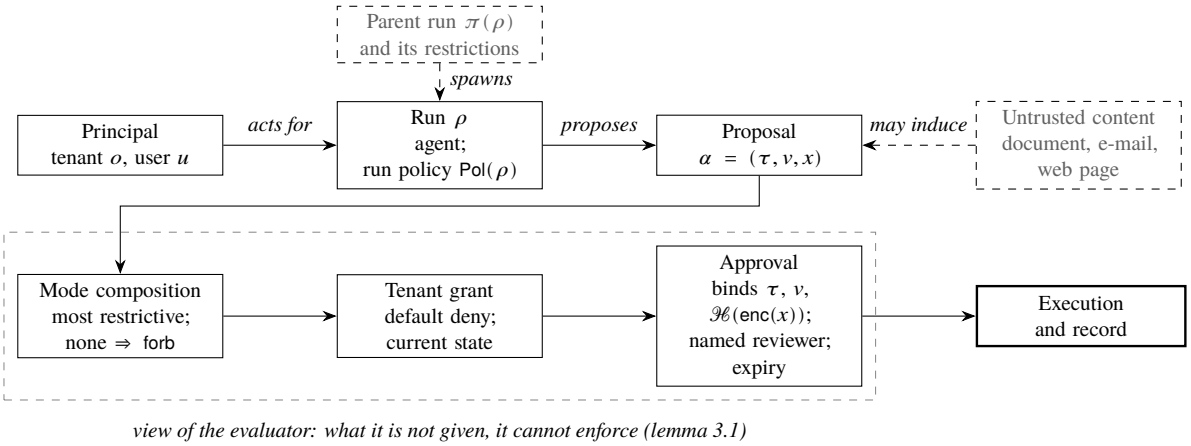

Figure 1: The path from a principal to an executed act. Solid boxes are the elements of the model; dashed boxes are facts on which norms N6 and N7 depend, which an evaluator can use only if they are carried to it.

The lemma is trivial. Its use is not. It turns a question about an evaluator's logic, which can always be made more elaborate, into a question about its inputs, which can be settled by reading an interface. No evaluator, however capable, enforces a norm that depends on a fact it is not given; adding a model to the evaluator changes nothing if the model receives the same view. Sections 5, 7 and 8 apply the lemma to separation of duties, delegation and provenance by exhibiting, for each, a pair of scenarios that a given view cannot tell apart. The BEN-02 benchmark calls the corresponding attack classes *structural* for any evaluator whose inputs omit the fact in question (LL-BEN-02).

3.2 Composition by restriction

Proposition 3.1 (Restriction composes)**.** *Let $\Sigma \subseteq \Sigma'$ be sets of policy sources and suppose some source in Σ applies to (α, ρ). Then $m_{\Sigma}(\alpha, \rho) \leq m_{\Sigma'}(\alpha, \rho)$. In particular, adding a source never widens authority, no source can grant what another forbids, and an act to which no source applies is forbidden.*

Proof. Both effective modes are maxima, over the applicable sources of Σ and of Σ' respectively. The second set contains the first and both are non-empty, so the maximum cannot decrease. The last claim is the first case of equation (1). □

Two remarks qualify the proposition. First, it requires a non-empty set: the composition of no sources is forb, the top of the order, which is the default-deny convention rather than a maximum. A composition that returned the bottom element for an empty set would be monotone in the same sense and would allow every act that no rule mentions. Second, composition is only as complete as the set of sources the evaluator consults. A restriction held by a source that is not consulted restricts nothing; section 7 shows that this is exactly how delegation can widen authority in a system whose composition is otherwise correct.

Lawie. The runtime takes the most restrictive of the capability modes of every source it consults [IMPLEMENTED]; an approval-gated mode needs an approval for the exact call [TESTED]. The composition of an empty set of modes is *forbidden*, and the decision schema rejects a decision that is allowed in mode *forbidden* or allowed in mode *requires approval* without approval evidence [TESTED]. The tenant grant decision is default-deny: without an active grant the call is refused [TESTED].

4 Approvals as content-addressed acts

An approval is worth what it binds. A reviewer who approves "send the engagement letter to the client" and thereby also authorizes "send the settlement offer to the counterparty" has approved nothing in particular. The approval must designate one act, and at execution time the evaluator must tell whether the act about to run is that act. Content addressing does this: the approval carries the digest of an encoding of the input, and the evaluator recomputes the digest from the input that will execute.

4.1 The binding result

Assumption 4.1 (Collision resistance)**.** *No efficient adversary finds byte strings $m \neq m'$ with $\mathcal{H}(m) = \mathcal{H}(m')$, except with negligible probability.*

Assumption 4.2 (Encoding)**.** *For every tool τ,* enc *is injective on X_τ (inputs the tool treats differently have different encodings) and deterministic (every process that computes* $\mathsf{enc}(x)$ *obtains the same bytes).*

Assumption 4.3 (Recomputation)**.** *The evaluator computes the digest from the input that will execute, or from an input from which the executed input is derived deterministically under the approved version, and never accepts a supplied digest in its place.*

Proposition 4.1 (Approvals cannot be reused)**.** *Under assumptions 4.1 to 4.3, let a be an approval granted for $\alpha = (\tau, v, x)$. If an evaluator that requires a to bind allows $\alpha' = (\tau', v', x')$ on the strength of a, then $\tau' = \tau$, $v' = v$, and either x' and x are the same input of τ or $(\mathsf{enc}(x), \mathsf{enc}(x'))$ is a collision of SHA-256. Hence reuse of an approval for a different input succeeds with negligible probability, even against an adversary who chooses both inputs.*

Proof. Allowing α' on the strength of a requires a binds α', so $\tau' = \tau$, $v' = v$ and $\mathcal{H}(\mathsf{enc}(x')) = h = \mathcal{H}(\mathsf{enc}(x))$, with h computed from x when a was created and $\mathcal{H}(\mathsf{enc}(x'))$ from the executing input (assumption 4.3). If x' and x are different inputs of τ, assumption 4.2 gives $\mathsf{enc}(x') \neq \mathsf{enc}(x)$, and the two encodings collide, which by assumption 4.1 an efficient adversary achieves with negligible probability. □

The adversary may choose both inputs: an agent, or a person steering it, proposes x for review and presents x' for execution. Second-preimage resistance, which protects only a target the adversary did not choose, would not suffice.

4.2 What the binding does not do

It does not prevent repetition. An approval that binds α binds it every time α is proposed before expiry; acts that must not be repeated, such as a payment or a filing, need the approval to be *consumed* atomically by the execution it authorizes, which is state, not cryptography. *It does not show understanding.* It ensures that what executes is what the reviewer was shown, not that the reviewer read it or knew why it was proposed (section 8). *It is only as good as the encoding.* If the encoding discards a distinction the tool makes, two different acts share an approval, a failure of safety. If the encoding depends on the computing process, the same act yields different digests in different processes, a failure of availability and accountability: a legitimate approval is refused, and a third party cannot recompute the digest in a record. A canonical serialization with a fixed key order, such as RFC 8785 [13], satisfies both halves of assumption 4.2 for JSON data. *Versions are part of the act,* because a tool's behaviour can change while its input stays the same.

4.3 Lawie

An approval in Lawie authorizes only the tool identity, its exact semantic version and the SHA-256 of the canonical current input, unless a requirement explicitly names another tool and version; a referenced digest can narrow but never replace the digest of the current input [TESTED]. The approval binds the input the agent proposed, a JSON value whose version-pinned validation is what executes, and the agent runtime and the core service compute the digest with the same function [IMPLEMENTED]. On JSON values the encoding is injective up to key order, the sign of zero and numbers outside the double range [ANALYTICAL]. An approval is usable only if approved, reviewed by a named reviewer after it was requested and before it is used, and unexpired; a malformed composite policy is invalid, never approved [TESTED]. Approval lifetimes lie between fixed bounds set at request, and the core service refuses to reuse an approval identifier for another run, tool, version, input or lifetime [IMPLEMENTED]. On the path that dispatches writes to external providers each approval is consumed once [IMPLEMENTED].

5 Separation of duties and quorum

Separation of duties requires that no single person both initiate and complete a sensitive act. It is central to commercial integrity policy [4], a constraint type in role-based access control [5] and an instance of the principle that critical operations need more than one key [2]. Quorum generalises it: k distinct persons from an eligible set must approve.

5.1 Separation of duties needs the requester

Proposition 5.1 (No separation of duties without the requester)**.** *Suppose* view *does not depend on the persons responsible for a request:* $\mathsf{view}(s) = \mathsf{view}(s')$ *whenever s and s' differ only in R. Let s be an authorized scenario in which an approval a reviewed by $r = \mathsf{rev}(a) \notin R(s)$ is relied on. Then every evaluator over* view *that allows s also allows the scenario s' that differs from s only in $R(s') = \{r\}$, in which the reviewer approved their own request.*

Proof. By hypothesis $\mathsf{view}(s') = \mathsf{view}(s)$. s satisfies N3 and s' violates it, since $\mathsf{rev}(a) = r \in R(s')$. Lemma 3.1 gives the claim. □

Corollary 5.1. *An evaluator whose view contains no requester identity cannot enforce separation of duties without refusing legitimately approved acts.*

In an agent system the agent is not a requester in the sense of the norm: it cannot bear responsibility, and recording which agent proposed an act identifies software, not an accountable person. $R(s)$ holds the user on whose behalf the run acts and, where runs are nested, the users of every ancestor run; the check must be made where the approval is relied on, so the lineage must travel with the request.

5.2 Quorum needs counting

Definition 5.1 (Quorum)**.** A *k-quorum over an eligible set $E \subseteq U$* holds for α at t if there is a set W of approvals, each valid at t and binding α, with $|\{\mathsf{rev}(a) : a \in W\}| \geq k$ and $\mathsf{rev}(a) \in E$ for every $a \in W$.

Proposition 5.2 (Unnamed requirements cannot require two persons)**.** *Let q be a requirement (definition 2.6) in which no reference names a reviewer. For every set G of approvals, every act and time, and every user r: if $G \models q$ then $G[r] \models q$, where $G[r]$ replaces every non-empty reviewer in G by r. Hence no such q that any set of approvals satisfies enforces a k-quorum for $k \geq 2$. A requirement that names reviewers can enforce a k-quorum over a fixed set E only by encoding the threshold over named reviewers (for example as an* anyOf *over every k-subset of E), and must be rewritten whenever E changes.*

Proof. By induction on q. For $\mathsf{single}(ref)$ with no reviewer named in *ref*, satisfaction depends on an approval's identifier, tool, version, digest, status and instants, and on its reviewer only through $\mathsf{rev}(a) \neq \bot$; replacing every non-empty reviewer by the same r leaves empty reviewers empty and preserves all of these. anyOf and allOf are Boolean combinations of their children. seq adds a condition on review instants, which $G[r]$ leaves unchanged. Since $G[r]$ has one reviewer, a requirement satisfied by it cannot imply two distinct reviewers. For the last claim, the same induction shows that replacing every non-empty reviewer whom q does not name by one user whom q does not name preserves satisfaction; distinct persons can therefore be required only among the reviewers q names, so a k-quorum over E must name members of E and express the threshold over them. A member added to E is not named, and their approvals cannot count toward distinctness until q is rewritten. □

The proposition separates two things that policy languages tend to merge: a conjunction of conditions and a count of persons. A requirement written "approval A and approval B" reads as two approvals, and a Boolean evaluator reads it as two predicates over the same pool. Law writes counts. Article 14(5) of the AI Act, as published, requires for the high-risk AI systems referred to in point 1(a) of its Annex III that "no action or decision is taken by the deployer on the basis of the identification resulting from the system unless that identification has been separately verified and confirmed by at least two natural persons with the necessary competence, training and authority" [14]. (Article 113 of Regulation (EU) 2024/1689, as amended by Regulation (EU) 2026/1744 [15], lists 2 December 2027 for the application of the section containing Article 14 to Annex III systems [EXTERNAL].) The provision concerns identification systems; the point here is structural. An evaluator under which one person's two approvals satisfy a two-approval requirement cannot implement a rule of that form, and neither can a policy author using its language, except by naming the persons.

6 Tenant isolation

A platform that serves several firms must ensure that an act requested in one tenant affects nothing in another. Norm N5 states the requirement for one act; the difficulty is "every tenant the act designates". An input can designate a tenant directly, by an identifier under some field name, or indirectly, by the identifier of a document, matter or account that belongs to a tenant.

Observation 6.1 (Name-based scans are incomplete)**.** *Let a scan inspect the values under a fixed set K of field names, at any depth. If a tool's input schema admits a tenant identifier under a name outside K, the scan passes an input that designates a foreign tenant under that name. Such a scan is complete only for tools whose schemas confine every tenant designation to names in K.*

The observation needs no proof; its use is in what it rules out. Designations are open-ended: new tools, nested objects and indirect identifiers keep adding names. Three designs avoid enumerating them, in increasing strength: a scan that follows declared *types* rather than names; tools that accept no tenant designation and take the tenant from the authenticated principal; and a data layer that binds every read and write to the session's tenant, so that a foreign identifier designates nothing the session can reach. This is the old distinction between designation and authority [1, 3]: in a capability system holding the designation is holding the authority; in a system that checks identifiers against a principal, every designation must be found before it can be checked.

Lawie. At the basis commit, Lawie's tenant scoping compares the requested tenant with the authenticated principal's, for administrators too [TESTED]. The agent orchestrator requires a signed identity assertion bound to the request's method, path and body and rejects tenant or user substitution [TESTED]. These describe implemented checks, not an assurance that the tenants of a deployed service are kept apart.

7 Delegation and attenuation

When an agent spawns another, the child acts for the same principal. The question is with how much authority. A planner restricted to read-only research may delegate a sub-task to a specialist that, by its own definition, can file documents. Unless the planner's restriction travels with the delegation, the specialist can file, and the planner has obtained, through the child, an act it was denied. This is the confused deputy [3]: the child is a deputy that holds the tenant's authority and uses it on behalf of a requester that did not hold it. Capability systems avoid the problem by construction, because a child receives exactly the capabilities passed to it and a capability can be passed on only with the same or fewer rights [1]; bearer credentials that carry their own restrictions apply the same idea to distributed systems [16].

7.1 The attenuation result

Let $\mathscr{A}$ be the set of acts. For a run ρ, $\mathsf{Pol}(\rho) \subseteq \mathscr{A}$ is the set of acts its own definition and run policy permit, and $\mathsf{B}(o) \subseteq \mathscr{A}$ the set that the tenant's grants permit. A root run holds $\mathsf{Auth}(\rho_0) = \mathsf{Pol}(\rho_0) \cap \mathsf{B}(o)$. A *delegation rule* D assigns a child its authority $\mathsf{Auth}(c) = D(\mathsf{Pol}(c), \mathsf{Auth}(p), \mathsf{B}(o))$ when $p \rightsquigarrow c$.

Definition 7.1 (Safe and complete delegation)**.** D is *safe* if always $\mathsf{Auth}(c) \subseteq \mathsf{Auth}(p)$ and $\mathsf{Auth}(c) \subseteq \mathsf{Pol}(c)$: the child exceeds neither its parent nor its own policy. D is *complete* if always $\mathsf{Auth}(c) \supseteq \mathsf{Auth}(p) \cap \mathsf{Pol}(c)$: the child is not denied what both its parent's authority and its own policy permit.

Proposition 7.1 (Attenuation is intersection)**.** *A delegation rule is safe and complete if and only if* $\mathsf{Auth}(c) = \mathsf{Pol}(c) \cap \mathsf{Auth}(p)$. *Under that rule every run in a tree holds no more than its root:* $\mathsf{Auth}(\rho) \subseteq \mathsf{Auth}(\rho_0) \subseteq \mathsf{B}(o)$*, at any depth. If instead* $\mathsf{Auth}(c) = \mathsf{Pol}(c) \cap \mathsf{B}(o)$*, the parent's authority does not enter, and whenever some act lies in* $\mathsf{Pol}(c) \cap \mathsf{B}(o)$ *but not in* $\mathsf{Auth}(p)$*, the child holds authority its parent lacks.*

Proof. Safety gives $\mathsf{Auth}(c) \subseteq \mathsf{Auth}(p) \cap \mathsf{Pol}(c)$ and completeness the reverse inclusion, so a safe and complete rule is the intersection; the intersection is plainly safe and complete. For the bound, $\mathsf{Auth}(c) \subseteq \mathsf{Auth}(p)$ at every edge, and induction along the path from the root gives $\mathsf{Auth}(\rho) \subseteq \mathsf{Auth}(\rho_0)$. For the last claim, an act in $\mathsf{Pol}(c) \cap \mathsf{B}(o)$ outside $\mathsf{Auth}(p)$ is in $\mathsf{Auth}(c)$ and not in $\mathsf{Auth}(p)$. □

In the terms of lemma 3.1, an evaluator that decides a child's calls without the parent's restrictions has a view in which a widening scenario and a permitted one coincide, so it cannot enforce N6. Nor can the child's own policy do the work: the parent's model writes the child's task, and untrusted content can steer that model (section 8). Only the intersection with the parent's authority bounds the child whatever its task says.

8 Provenance of a request

An agent's proposal depends on what its principal asked for and on what it read while doing so: $\alpha = f(i, c)$ for instruction i and content c. A proposal is *principal-induced* if the instruction warrants it and *content-induced* if it was produced because of the content, such as a clause in a counterparty's e-mail asking for the draft to be sent "to the address below" or an instruction hidden in a retrieved document. Indirect prompt injection is the practice of writing such content [6], and simple adversarial inputs suffice to redirect a model's goal [7]. Provenance is in general a counterfactual question (would the act have been proposed without c?) and is not computable from the act alone. The model takes the label as part of the scenario; the result concerns what an evaluator can do without it.

Proposition 8.1 (Provenance blindness)**.** *Suppose* $\mathsf{view}(s) = \mathsf{view}(s')$ *whenever s and s' differ only in* prov. *Then for every principal-induced scenario that an evaluator over* view *allows, it also allows the content-induced scenario with the same act, run, approvals and time.*

Proof. The two scenarios have the same view; the first satisfies N7 and the second violates it. Apply lemma 3.1. □

The content-induced twin of a legitimate scenario is realised by any adversary who controls content the agent reads and can steer the model to propose an act the principal might also have asked for, which is what the injection literature demonstrates [6]. If p is the rate at which a model proposes injected acts and q the rate at which the layer allows a proposed act, then $\Pr(\text{an injected act executes}) = p \cdot q$, and a provenance-blind layer decides an injected act exactly as it decides the same act requested. Benchmarks such as AgentDojo and InjecAgent estimate p for given models and defences [17, 18]; BEN-02 characterises a layer's decisions per class (LL-BEN-02); it does not estimate a deployment rate. For acts it would allow on request, a provenance-blind layer contributes nothing against injection, and the burden falls on p, a property of a model that the operator does not control.

Identity is often mistaken for provenance. *Identity is not intent*: a signed assertion proves which principal's session issued a request, and every content-induced request carries a valid identity because the principal's own agent issued it. Three designs give the evaluator what proposition 8.1 says it lacks, each at a cost:

1. *Information-flow labels*: mark data derived from untrusted content, propagate the mark, and refuse or escalate consequential acts whose arguments carry it. *Cost:* labels spread, and escalations with them.
2. *Approval with provenance shown*: route consequential acts to a bound approval that displays the content that preceded the proposal. *Cost:* attention; an approval that shows the act but not its origin asks for a blind judgment.
3. *Capability reduction while reading*: fix the acts a run may perform before it reads untrusted content. *Cost:* tasks that depend on what is read need a new run and a new authorization.

9 Risks specific to many agents

Systems of agents add risks that arise from composition. Two of them reduce to propositions applied along a chain.

Diffusion of responsibility. With several agents between a principal and an act, "who authorized this?" has several plausible answers: the user who started the root run, the author of the agent definition, the administrator who granted the tool, the reviewer of the input. Increasingly agentic systems open such gaps in responsibility and erode accountability [8]. Only a record naming the principal of every run in the lineage, every approval relied on and every reviewer answers the question; a record that names the agent answers a different one.

Laundering. A chain of delegations without intersection lets a descendant use authority an ancestor was denied (proposition 7.1). A child's result returns to its parent as content, so a parent's next proposal can be content-induced at one remove; provenance must cross agent boundaries, and a label dropped at the first hop protects nothing. Each hop is unremarkable on its own, which is why both widenings are hard to see in review.

Approval fatigue. More agents produce more approval requests; binding ensures that a reviewer approves what will execute, not that the reviewer attends to it. We have no measurement of reviewer behaviour.

Readiness is not authorization. A checklist of an agent's readiness is a release decision about software and confers no authority at run time. An agent's authority at run time must come from the mechanisms of sections 3, 4 and 6, not from a release gate.

Table 1: Controls implemented in Lawie at the basis commit. The evidence column gives the register's classes for the statements in the row.

Control family	What it does	Evidence
Composition and grants	Effective authority is the most restrictive mode of every consulted source, and no applicable mode means refusal; tool calls need an active tenant grant within its ceilings, checked against current state on every call	[TESTED] [TESTED] [IMPLEMENTED] [TESTED] [IMPLEMENTED]
Approvals	Approvals bound to the exact action and checked for validity	[TESTED] [IMPLEMENTED] [TESTED] [IMPLEMENTED] [IMPLEMENTED] [IMPLEMENTED] [TESTED]
Identity and tenant	Signed request identity bound to method, path and body; requested tenant compared with the authenticated tenant	[TESTED] [TESTED] [TESTED]

Failure of the authority service. Many agents multiply calls to the authorization service and the dependence on its availability. The safe answer when it cannot decide is refusal.

10 Lawie's controls and its BEN-02 outcome

Table 1 summarises the controls Lawie implements at the basis commit. Each row describes checks in the source; it does not establish that every deployed path calls them, which cannot be established for a system that is not deployed [ABSENT].

10.1 What BEN-02 measured

BEN-02 ran three of Lawie's authorization functions, the requested-tenant check, a check of tenant identifiers inside the tool input and the approval evaluator [IMPLEMENTED], on 552 synthetic scenarios from four consequential tools, six inputs each and 23 templates, beside permissive, allow-list and unbound-approval reference systems. Lawie's functions denied none of the 96 legitimate requests, refused every scenario of some attack classes and allowed every scenario of others [MEASURED]. Which classes these are is recorded in restricted engineering records pending remediation. LL-BEN-02 reports the benchmark's design, the analysis of which classes any evaluator can refuse given its inputs, and the reference systems' results class by class.

> **What the measurements do not show.** BEN-02 is component-level and synthetic. It did not exercise the core service's database-backed grant decision, the agent admission ceiling, the runtime policy engine or any end-to-end workflow; their behaviour is described here from source and tests. A refusal means that the function refused, not that a deployed path calls the function on every route. No production or customer data was used, and nothing in this section is a measurement of a deployed system.

11 Requirements and residual risks

11.1 Ten requirements

The propositions yield requirements for any system in which agents propose consequential acts. Each states its cost. They are principles derived from the model, not a work plan.

R1 Deny by default; compose by restriction (proposition 3.1). *Cost:* work is refused until granted.

R2 Mediate every call at the point of use against current state, and refuse when authority cannot be established. *Cost:* a lookup per call; every act depends on the authority service.

R3 **Bind approvals to tool, version and the digest of an injective, deterministic encoding, recomputed from the executing input** (proposition 4.1). *Cost:* any change needs a new approval.

R4 **Consume approvals of acts that must not be repeated, atomically with their execution.** *Cost:* state and concurrency control.

R5 **Record the persons responsible for a request, across the lineage of runs, and refuse when a reviewer is among them** (proposition 5.1). *Cost:* small teams need outside reviewers.

R6 **Count distinct persons from an eligible set defined by role, at the time of use** (proposition 5.2). *Cost:* more approvers, slower decisions.

R7 **Take the tenant from the authenticated principal, enforce it in the data layer, and check remaining designations by type** (observation 6.1). *Cost:* schemas must declare tenant fields.

R8 **Intersect a child's authority with its parent's; carry restrictions and lineage to every descendant** (proposition 7.1). *Cost:* children can do less.

R9 **Carry provenance from content to proposals; refuse or escalate consequential acts that untrusted content influenced; show provenance to reviewers** (proposition 8.1). *Cost:* labels spread, escalations rise.

R10 **Record each decision with the lineage of principals, the approvals and reviewers relied on, and the model, prompt and policy versions, in a log whose head an independent party witnesses.** *Cost:* witnessing infrastructure and care over disclosure.

11.2 Risks that remain when all ten are met

1. *A principal who intends harm.* Authorization limits who may act, not what they intend.
2. *Colluding reviewers.* Separation of duties and quorum assume independent decisions; an evaluator can count identities, not verify independence.
3. *Approval without attention.* Binding makes the reviewer approve what will execute, not read it; a reviewer who approves everything turns a bound approval into a formality with a signature.
4. *Imperfect provenance.* Influence is counterfactual and not computable in general: labels spread too far or miss influence carried by paraphrase.
5. *Authorized but wrong.* The right person can approve the wrong text. Authorization does not establish correctness, which the reliability papers treat (LL-REL-01, LL-BEN-01).
6. *Stolen identity.* The system acts on a stolen credential as on the real one until it is revoked; current-state checks shorten the window, they do not close it.
7. *The operator.* Whoever operates the evaluator controls its decisions and records; LL-SEC-01 sets out what must be trusted to the operator and what a third party can verify without that trust.

12 Limitations and conclusion

Limitations. The model treats one decision at one instant and leaves out concurrency, the interval between a check and the use it authorizes, partial failure and the economics of review. Norms N1 to N7 are Lawie Labs' statement of what authorization requires; an institution may require less or more, and the lemma carries over to any norm of the same structure. Statements about Lawie describe source at the basis commit: component tests and an offline benchmark show what functions decide on stated inputs, not what a deployed system does. Nothing described here is in production [BLOCKED], no statement has been reviewed by an independent auditor or expert [ABSENT], and this paper has had no independent peer review. This paper reports research measurements at a stated code basis. It is not a security assurance, a compliance statement or legal advice, and no independent security, privacy or legal review of it has taken place. The application date quoted in section 5 is a research label read from the official text; it has not been checked by a qualified lawyer.

Conclusion. An agent's competence is the wrong place to look for its authority. Authority is a relation between persons, institutions and rules, and a system that executes acts for others must be able to show that relation for every act. Two of the results here are positive: restrictive composition and content-addressed approval are sound under explicit assumptions. Four are negative: separation of duties, quorum, attenuation and provenance cannot be enforced by an evaluator that is not given the requester, does not count persons, does not see the parent or does not know where a request came from. These results hold for any evaluator. The negative results are the more useful. They name the facts a governed multi-agent system must record at their origin and carry to the point of decision. A system that does not carry them cannot be made safe by improving the model that proposes its acts.

References

[1] Jack B. Dennis and Earl C. Van Horn. Programming Semantics for Multiprogrammed Computations. *Communications of the ACM*, 9(3):143–155, 1966. doi: 10.1145/365230.365252.

[2] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

[3] Norm Hardy. The Confused Deputy (or Why Capabilities Might Have Been Invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38, 1988. doi: 10.1145/54289.871709.

[4] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, 1987. doi: 10.1109/SP.1987.10001.

[5] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996. doi: 10.1109/2.485845.

[6] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90, 2023. doi: 10.1145/3605764.3623985.

[7] Fábio Perez and Ian Ribeiro. Ignore Previous Prompt: Attack Techniques for Language Models, 2022. arXiv:2211.09527.

[8] Alan Chan, Rebecca Salganik, Alva Markelius, Chris Pang, Nitarshan Rajkumar, et al. Harms from Increasingly Agentic Algorithmic Systems. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 651–666, 2023. doi: 10.1145/3593013.3594033.

[9] OWASP Foundation. OWASP Top 10 for LLM Applications 2025, 2024. URL https://genai.owasp.org/llm-top-10/.

[10] H. L. A. Hart. *The Concept of Law*. Clarendon Press, Oxford, 1961.

[11] Joseph Raz. *The Authority of Law: Essays on Law and Morality*. Clarendon Press, Oxford, 1979.

[12] Lawrence Lessig. *Code and Other Laws of Cyberspace*. Basic Books, New York, 1999.

[13] Anders Rundgren, Bret Jordan, and Samuel Erdtman. JSON Canonicalization Scheme (JCS). RFC 8785, Internet Engineering Task Force, 2020.

[14] European Parliament and Council of the European Union. Regulation (EU) 2024/1689 of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L, 2024/1689, 12.7.2024, 2024. URL https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32024R1689.

[15] European Parliament and Council of the European Union. Regulation (EU) 2026/1744 of 8 July 2026 amending Regulations (EU) 2024/1689, (EU) 2018/1139 and (EU) 2023/1230 as regards the simplification of the implementation of harmonised rules on artificial intelligence (Digital Omnibus on AI). Official Journal of the European Union, L, 24.7.2026, 2026. URL https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32026R1744.

[16] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Proceedings of the 2014 Network and Distributed System Security Symposium*, 2014. doi: 10.14722/ndss.2014.23212.

[17] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems 37 (Datasets and Benchmarks Track)*, 2024.

[18] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506, 2024. doi: 10.18653/v1/2024.findings-acl.624.