Zemoso

Beyond identity: A preventive architecture for AI agent governance

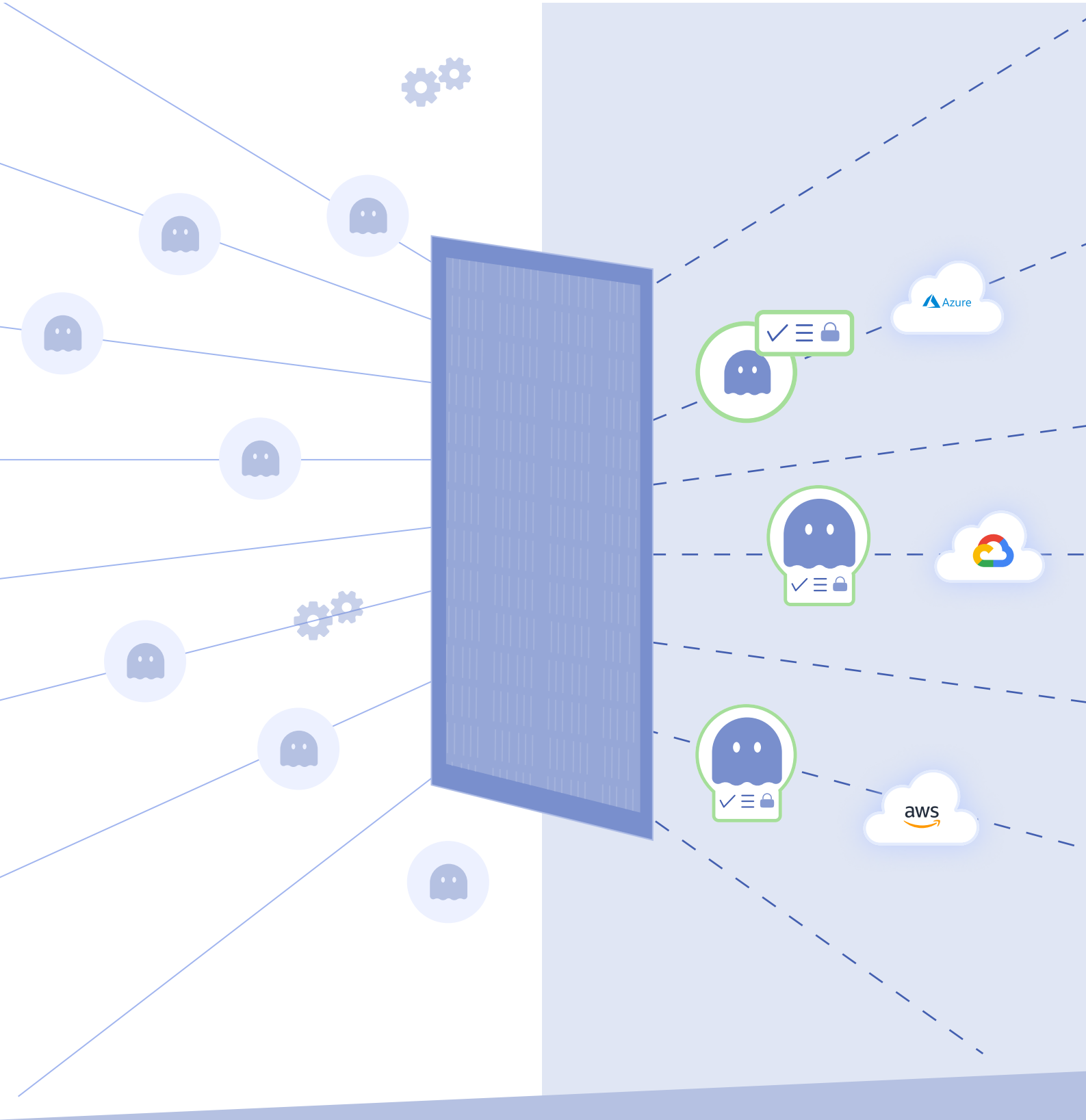


Table of contents

Introduction 1

The new problem is delegated autonomy 2

Where conventional governance quietly fails 3

The key design principle: the model is not the security boundary 4

How to make a forbidden action architecturally impossible 5

What secure context propagation should look like 7

When a user needs an answer—not access to the system of record 8

- Three independent gates
- A usable PLM-assisted workflow

A practical example: a tenant-safe refund 11

Standards are converging—but the architecture still matters 12

Intent should be recorded as evidence, not trusted as authority 13

Governing the digital workforce 13

For nearly two decades, enterprise security has revolved around a simple assumption: **humans make decisions, software executes them.**

Identity and Access Management (IAM), Zero Trust, Privileged Access Management (PAM), and cloud governance frameworks were all designed around this model. Users authenticate, policies determine what they are permitted to do, and systems record what happened.

Artificial Intelligence is changing that assumption.

Today's AI agents are no longer passive assistants or deterministic scripts. They interpret objectives, reason through alternatives, invoke tools, collaborate with other agents, and orchestrate work across multiple cloud platforms with minimal human intervention. They represent a new class of enterprise actors, one capable of exercising delegated autonomy.

This evolution introduces a governance challenge that traditional cloud security models were never designed to solve.

The problem is not simply that AI agents possess identities. Increasingly, organizations are recognizing that non-human identities require lifecycle management, governance, and policy enforcement comparable to human users. Gartner, Forrester, NIST, and the Cloud Security Alliance all point toward this shift from different perspectives.

The deeper challenge lies elsewhere.

Modern AI agents are capable of making decisions before enterprise policy has been evaluated. They can plan actions, sequence tools, interact across cloud boundaries, and operate through layers of orchestration that gradually obscure who initiated the request, what authority was delegated, and which policies should govern execution.

In other words, **autonomy, not intelligence, is redefining enterprise governance.**

Organizations need more than stronger identity management for AI. They need a preventive governance architecture that deliberately separates planning, authorization, and execution.

Instead of allowing AI models to carry standing authority, the architecture proposed in this paper treats the model as a planning engine. Trusted infrastructure evaluates intent against enterprise policy, issues narrowly scoped and short-lived authority where

appropriate, and ensures that protected resources independently enforce every decision.

This approach transforms AI governance from a reactive activity, reconstructing events after execution, to a preventive one, where policy boundaries exist independently of the model's behavior.

The result is a governance model that preserves attribution, authority, context, and accountability across autonomous workflows, while remaining aligned with emerging standards in workload identity, delegated authorization, and AI risk management.

The new problem is delegated autonomy

Enterprise security has never been exclusively human. Service accounts, workload identities, API clients, certificates, bots, and CI/CD runners have existed for decades. The real change is that an agent is not merely executing a predetermined sequence. It can interpret an objective, select a tool, combine information, delegate work, and adapt its next action—all at machine speed.

That creates a compound identity problem. An agentic action can involve at least four distinct actors: the human or system that initiated the task, the agent workload that planned it, the application or workflow within which it is operating, and the tool or downstream service that executes it. Collapsing those actors into one shared API key destroys both least privilege and accountability.

In a multi-cloud estate, the problem becomes harder because identity, authorization, logging, and resource semantics differ across AWS, Microsoft Azure, Google Cloud, SaaS platforms, and internal systems. The risk is not simply that an agent may make a poor decision. It is that the architecture may accidentally give the model a path around the organization's real security boundaries.

Where conventional governance quietly fails

1. The attribution gap

Enterprise security has never been exclusively human. Service accounts, workload identities, API clients, certificates, bots, and CI/CD runners have existed for decades. The real change is that an agent is not merely executing a predetermined sequence. It can interpret an objective, select a tool, combine information, delegate work, and adapt its next action—all at machine speed.

2. The authority and tenancy gap

A user may be permitted to read data for Tenant A, while the agent runtime itself can reach data for every tenant. If tenant_id is merely a prompt instruction or a request field supplied by the model, it is not a security control. Prompt injection, a faulty tool call, or a programming error can cross the boundary.

3. The context gap

Authorization context is often lost or widened as a request moves from the user interface to an orchestrator, from one agent to another, and then to an MCP server or cloud API. Forwarding a bearer token unchanged does not solve this problem; it can create token theft, confused-deputy, and audience-confusion risks.

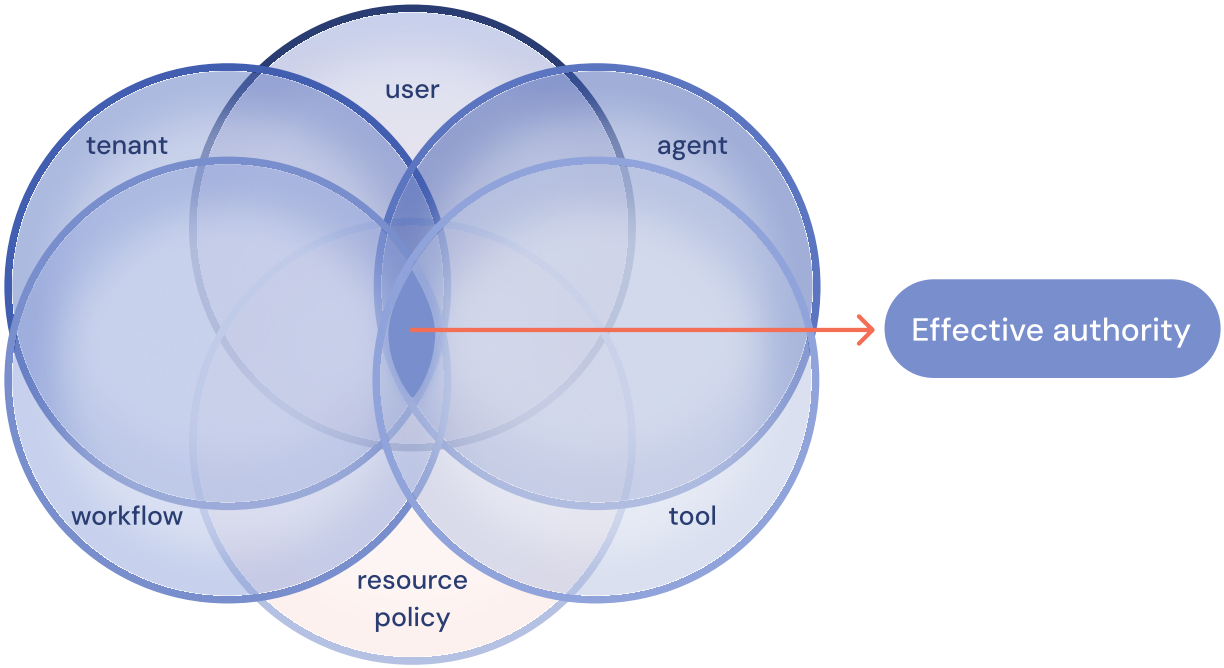
4. The accountability gap

Traditional audit records explain what happened. Agent governance must also preserve the decision context: the initiating principal, objective, agent and model version, policy decision, tools considered and invoked, data classifications touched, approval state, and outcome. This is not the same as storing private chain-of-thought, which may be unavailable, unreliable as an explanation, and unsafe to retain.

The key design principle: the model is not the security boundary

An agent can be instructed to obey a policy, but instructions are not enforcement. The model should be treated as an untrusted planner that proposes actions. A trusted control plane decides whether those actions are permitted, and the tool or resource enforces that decision again.

Effective authority = user ∩ tenant ∩ agent ∩ workflow ∩ tool ∩ resource policy

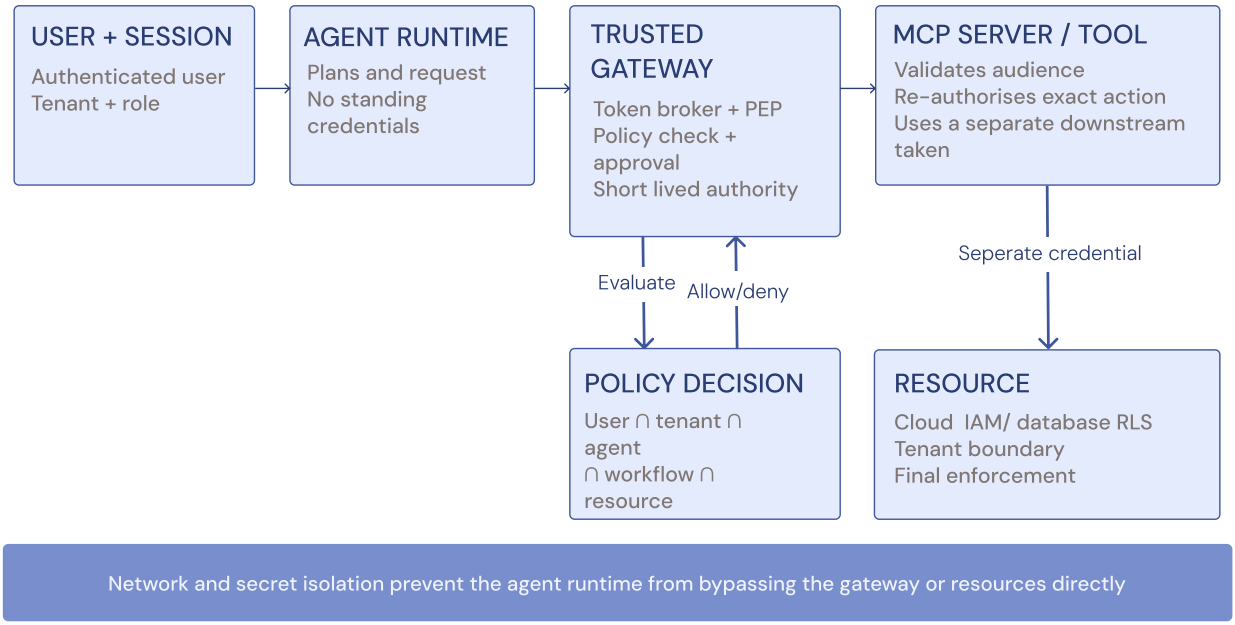


For ordinary delegated actions, the intersection matters. The agent runtime should not receive broader direct authority than the initiating user, and the workflow, tool, resource, environment, purpose, and time window may impose narrower limits. Delegation should be monotonically decreasing: every hop can preserve or reduce authority, never expand it without a new, explicit authorization event.

There is, however, an important enterprise exception. A user may be entitled to invoke an approved function over protected information without being entitled to browse the underlying records. In that case, a separate trusted service—not the general-purpose agent—uses its own institutional authority to perform a controlled computation. A second policy then determines what result, if any, may be disclosed to the requester.

A preventive architecture for agentic actions

The model proposes. Trusted components carry and enforce authotity



How to make a forbidden action architecturally impossible

No distributed system is literally unbreakable. But the normal execution path can make a policy violation impossible for the agent—even if the model is manipulated—by removing both the credential and the network path needed to perform it.

1. **Separate planning from execution.** Run model reasoning in a sandbox with no cloud credentials, database passwords, or unrestricted network access. The agent emits a structured action request; a separate executor validates and performs it.

2. **Authenticate every principal.** Preserve the initiating user or service identity, the agent/workload identity, and the application or workflow identity as distinct claims. Use short-lived workload identity rather than embedded secrets.

3. **Mint authority in a trusted component.** The agent must not be able to set tenant, role, scope, approval, or delegation claims. A gateway or token service derives them from authenticated session state and policy, then issues a short-lived, audience-bound credential for a specific tool or resource.

4. **Bind authority to the call.** Prefer proof-of-possession or mutual TLS where feasible, along with narrow audience, resource, action, tenant, purpose, expiry, and correlation claims. A signed bearer token protects integrity but can still be stolen and replayed.

5. **Exchange at every trust boundary.** Do not pass a user token unchanged through an MCP server to a downstream API. Validate the inbound token, re-authorize the exact operation, and obtain a separate, down-scoped downstream credential.

6. **Enforce at the tool and the resource.** An MCP server should be a policy enforcement point, not a transparent proxy. The database should still apply row-level security or tenant-partitioned credentials; cloud IAM should still constrain resource and action. A compromised orchestrator should not be enough to cross tenancy.

7. **Constrain egress and discovery.** Allow-list tool endpoints, prevent arbitrary HTTP calls, isolate local MCP processes, and keep secrets outside the model's memory and prompt. If the agent cannot reach an unapproved endpoint, prompt injection cannot turn that endpoint into a side door.

8. **Use approvals as policy state.** High-impact, irreversible, novel, or cross-boundary actions should require a cryptographically attributable approval that is narrow in scope and expires. "Human in the loop" should not mean a generic click that grants a reusable admin session.

What secure context propagation should look like

The context sent to a tool should not be a collection of headers copied from the front end, and it should never be constructed by the model. It should be a verifiable authorization envelope created by a trusted issuer. Depending on the architecture, that may be an OAuth access token, a token-exchange result, a transaction token, a capability, or an opaque handle resolved by the receiving service.

At minimum, the receiving tool needs enough trusted information to answer five questions:

Who?

The initiating subject, the acting workload, and—when applicable—the delegating agent or workflow.

Where?

The tenant, customer account, project, data partition, cloud account, and environment.

What?

The exact action, target resource, data classification, and permitted field or row scope.

Why and under what conditions?

The tenant, customer account, project, data partition, cloud account, and environment.

How is it traced?

A transaction and parent-delegation identifier that survives agent-to-agent and tool-to-tool hops without allowing downstream services to rewrite history.

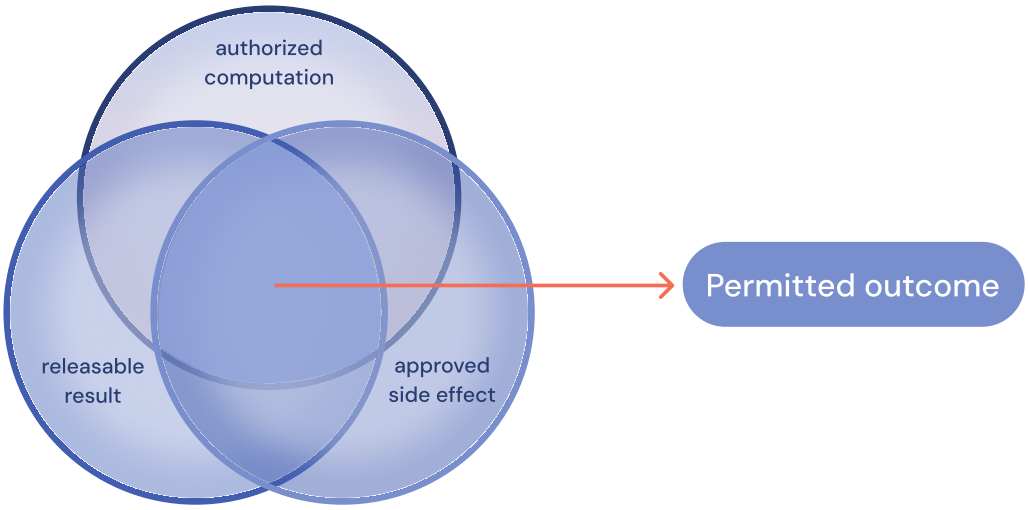
When a user needs an answer—not access to the system of record

Some of the most valuable enterprise information does not live in a simple application database. It lives in critical systems of record such as product lifecycle management (PLM), enterprise resource planning, manufacturing execution, clinical, financial, and engineering platforms. A PLM platform may serve as a source of product information

across the lifecycle. Depending on the implementation, that can include product structures and bills of material, revisions, CAD datasets, requirements, change records, manufacturing information, supplier artifacts, simulation evidence, and regulatory classifications. These systems contain some of the company's most consequential intellectual property.

A support engineer may nevertheless need to answer a legitimate vendor question: "Can Part P-7742 be manufactured using Alloy X instead of the specified Alloy Y?" Denying the question because the engineer cannot browse the entire PLM system is secure but unusable. Giving the engineer—or a general-purpose agent—a broad PLM service account is usable but unsafe.

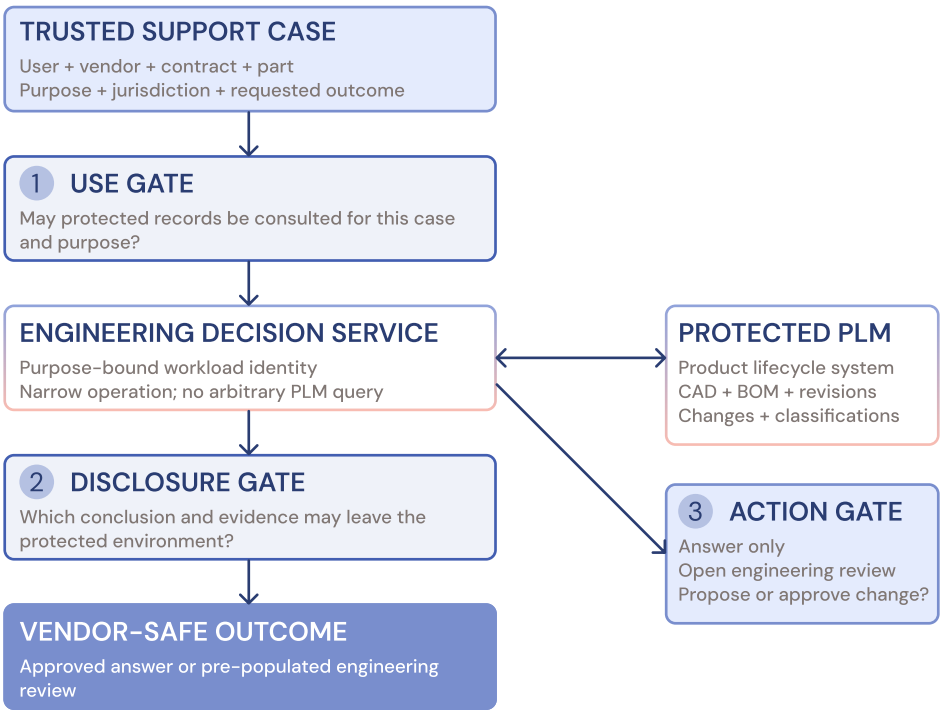
Permitted outcome = authorized computation ∩ releasable result ∩ approved side effect



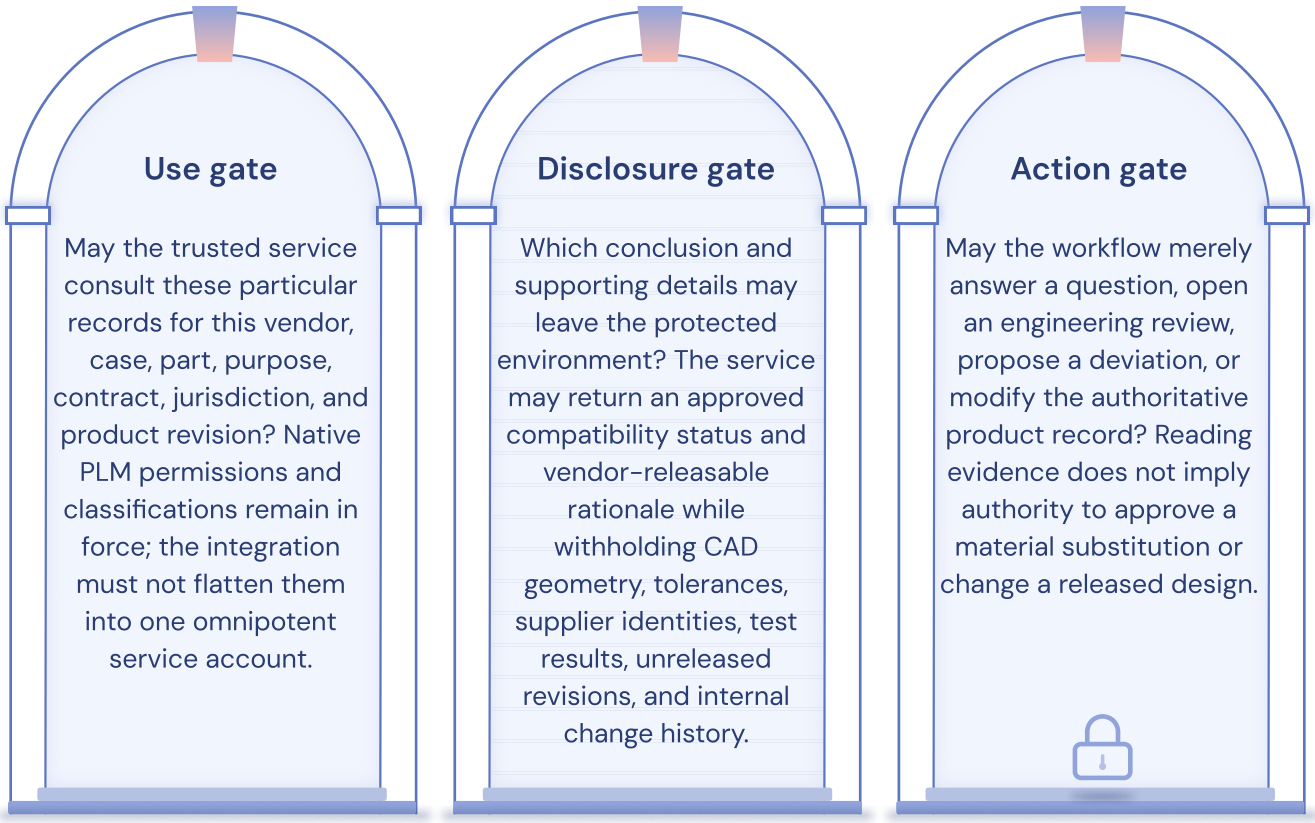
The better design is controlled computation with controlled disclosure. The user is authorized to invoke a narrow engineering decision service, not to read the source repository. The service can consult protected records under its own workload identity, while the agent receives only the minimum result needed to continue the support workflow.

Controlled computation over protected product IP

The requester receives an authorized outcome-not general access to the system of record.



Three independent gates



A usable PLM-assisted workflow

1. The support case establishes the vendor, contract, product, part, question, requester, jurisdiction, and permitted purpose. These facts come from trusted application state—not from the model's prompt.

2. The agent calls a narrow operation such as evaluate_material_substitution(part=P-7742, proposed_material=Alloy-X). It cannot download arbitrary datasets, enumerate neighboring products, or issue a general PLM query.

3. A trusted engineering service verifies vendor entitlement and resolves the correct released revision. It retrieves only the records needed for the evaluation using a purpose-bound workload identity.

4. Deterministic rules, approved calculations, and retrieved evidence produce a structured result. The general agent need not see the raw CAD files or complete bill of material.

5. An output policy returns only vendor-releasable information. A previously approved alternative may be answered automatically; a novel, safety-critical, export-controlled, or uncertain substitution is routed into the formal engineering-change workflow.

6. The response identifies its source revisions, limitations, approval state, and expiry. The audit trail records which protected objects were consulted and exactly what information crossed the boundary.

The secure path must also be the easiest path. Instead of responding "access denied," the agent should provide the permitted conclusion, explain what cannot be disclosed, and—when necessary—open a pre-populated engineering review. Controls that repeatedly block legitimate work without offering a usable route encourage copied files, shared credentials, shadow integrations, and unsanctioned AI tools.

Narrow tools are not automatically safe. Repeated questions can reconstruct a design one fact at a time. Query budgets, rate limits, detection of boundary-probing questions, cumulative disclosure tracking, and restrictions on bulk comparison are needed to address this mosaic risk.

A practical example: a tenant-safe refund

Suppose a support specialist assigned to Merchant A receives a legitimate request to refund $75 for Order 1042. A fragile design gives the AI agent a payment-platform administrator credential and trusts the merchant_id, order number, and amount supplied in its tool call. A preventive design works differently:

1. The specialist authenticates. The support application resolves their assignment to Merchant A, the customer case they are handling, and their $100 refund ceiling.

2. The agent proposes get_order(order=1042) followed by refund_order(order=1042, amount=75). It cannot supply an authoritative merchant identity or enlarge the operator's refund limit.

3. The trusted gateway evaluates the operator's merchant assignment, the order's association with the case, the agent's capabilities, the refund limit, and the risk of the action.

4. It issues a short-lived credential usable only by the payments MCP server, for one refund on Merchant A's Order 1042, capped at $75 and bound to an idempotency key and approval reference.

5. The MCP server derives Merchant A from the credential, verifies that the order belongs to that merchant and remains refundable, and obtains a separate downstream payment credential restricted to the same merchant and action.

6. The payment service performs its own final checks, and the user, agent, policy decision, approval, downstream transaction, and outcome are recorded as one traceable chain.

If an untrusted customer message tells the model to ignore its instructions and refund a $5,000 order belonging to Merchant B, the request fails at the gateway, the MCP server, and the payment service. The prompt is no longer the boundary.

Standards are converging—but the architecture still matters

Several initiatives address parts of this problem, but none is a complete agent-governance architecture.

Initiative	What it contributes—and what it does not
W3C AI Agent Protocol Community Group	Explores cross-domain agent discovery, identity, capability exchange, and collaboration. It is a useful direction of travel, but a Community Group is exploratory work—not a W3C Recommendation or a production security profile.
MCP authorization	Uses OAuth-based authorization for HTTP transports and requires audience validation; current security guidance forbids token passthrough. This secures the MCP transport boundary, but application-level tenant and object authorization still belongs in the MCP server and downstream resource.
IETF Transaction Tokens	A developing approach for preserving user, workload, and authorization context across a call chain within a trust domain. It closely matches the context-propagation problem, but it remains an Internet-Draft as of July 2026.
IETF WIMSE and SPIFFE/SPIRE	Address workload identity across heterogeneous and multi-cloud environments. They help authenticate the software making a call; they do not by themselves decide whether a particular user-backed action on a tenant resource is allowed.
OpenID AuthZEN	Standardizes communication between policy enforcement points and policy decision points. It can make distributed authorization more interoperable, while the organization still defines the policy model and enforcement locations.
NIST NCCoE software and AI-agent identity project	Applies established identity standards and practices to agents that take actions. It is important evidence that agent identity is becoming a mainstream security concern; its project material should be tracked according to its publication status.

The important distinction is authentication versus authorization. SPIFFE, workload credentials, or an agent identity registry can establish which software is calling. OAuth scopes and MCP authorization can constrain a transport. Neither automatically proves that this particular user, in this tenant, for this purpose, may perform this particular action on this object. That decision still requires policy and enforcement close to the resource.

Intent should be recorded as evidence, not trusted as authority

Organizations do need more than activity logs. They should preserve the user request or approved objective, the structured plan and tool arguments, retrieved evidence identifiers, policy inputs and decision, approval record, model and agent versions, result, and side effects. These records support investigation, testing, and accountability.

But an agent's stated intent must not authorize its action. A model can misunderstand a goal, be manipulated by untrusted content, or produce a plausible explanation after the fact. Intent is evidence for audit and risk evaluation; authority comes from authenticated principals, trusted context, policy decisions, and enforcement.

Governing the digital workforce

AI agents are becoming a distinct class of enterprise actor: not human users, not conventional applications, but delegated software capable of adaptive decisions and tool use. They require lifecycle governance—registration, ownership, testing, versioning, authorization, monitoring, suspension, and decommissioning—but lifecycle controls alone are not enough.

> *The safest agent is not merely one that has been told where the boundary lies. It is one that cannot obtain the credential or execution path needed to cross it.*
>
> - Sudhir Rao, Co-founder

The competitive advantage will not come from deploying the largest number of agents. It will come from giving each agent the minimum, verifiable authority required for one task —and being able to prove that every boundary was enforced before the action occurred.

The organizations that succeed will be able to answer:

- ✓ Who initiated the action ?
- ✓ Which agent and workload acted ?
- ✓ What authority was delegated ?
- ✓ Which tenant and resource boundary applied ?
- ✓ Why did policy allow it ?
- ✓ Where was the decision enforced ?

A modern security infrastructure does not constraint AI intelligence but reliably governs its agency. By building preventative boundaries directly into the multi-cloud infrastructure, enterprises ensure that the models grow more autonomous but their systems remain fundamentally safe by design.