

Designing agent authority boundaries

A practical architecture for deciding which actions an AI agent may take, for whom, and under what conditions.

Turn agent permissions into an enforceable action contract. Separate a model's suggestion from a resource owner's authorization, test the boundary, and make delegation and revocation reviewable.

FOR Security leaders, platform architects and agent engineering teams

Softlander Consultancy FZE LLC | 2026-09-16 | v1.0

The architecture and worked scenario are illustrative design tools, not results from a client deployment or a security certification.

In this paper

Start with the action, not the agent	2
Build an action contract	3
Put enforcement where the consequence occurs	4
Design approvals that people can understand	5
Delegation needs an expiry path	6
Choose controls the integration can support	7
Worked example: supplier changes without silent authority expansion	8
Test the boundary, not just the happy path	9
References	11

Start with the action, not the agent

An assistant that recommends a supplier and an agent that changes a supplier's payment destination create different security questions. Their chat interfaces may look identical. Their authority should not. A useful review therefore begins with the irreversible action and works backwards: which system can perform it, which identity permits it, which business rule constrains it, and which person owns the consequences?

This paper develops an action-contract approach for tool-using enterprise agents. The contract connects a legitimate business purpose to a narrowly specified operation. It records the initiating person or service, the resource, the permitted change, the approval requirement, the time boundary and the evidence needed afterwards. The contract is enforced at the tool or resource boundary; a model's own explanation is evidence to inspect, not permission to proceed.

OWASP describes excessive agency through unnecessary functionality, excessive permissions and excessive autonomy. These are separate design problems: removing an unused tool, narrowing its credentials and adding an approval gate address different exposures. [1] The practical consequence is that reviewing an agent's prompt is insufficient. Review its available operations, execution identity and decision rights together.

Imagine that the same agent can search documents, draft a purchase request and submit an order. A single "procurement assistant" role hides three permission decisions. Search may be bounded by document access; drafting may create no external commitment; ordering

may consume an approved budget. Each action should have its own checks, rather than inheriting authority from the overall product name.

The approach fits systems where an organization can control tool adapters, gateways or downstream permissions. It is less directly applicable to a closed external product that provides no action-level controls or evidence. In that case, the buying decision may be to restrict the product to advice, choose a different integration, or require a human to execute changes in the original system. A polished interface does not remove that architectural constraint.

Build an action contract

An action contract should be short enough to review beside an API operation. Begin with a sentence: "For this approved request, this identity may perform this operation on this resource until this condition expires." Then specify the boundaries that the sentence cannot express. Avoid a broad list of aspirational principles that an implementation team cannot translate into checks.

The actor field distinguishes the end user, the agent workload and any downstream service identity. A shared service account may simplify integration, but it can obscure whose business authority is being exercised. Record the initiator even when the downstream system cannot represent delegation directly. If the mapping is lossy, make that a documented restriction and compensate with a tighter operation or human execution step.

The resource field needs more than a system name. "Access the CRM" is broad; "read these account records for this assigned support case" is reviewable. Define tenant, object, field and operation where the platform allows them. The business-purpose field explains why this access exists, but purpose text alone does not enforce it. Connect purpose to a request, case or approved workflow that the enforcement point can verify.

The constraint field covers values and circumstances: permitted destinations, maximum transaction size, allowable state transitions, working scope and excluded data. The approval field identifies both who can approve and what they are approving. Approval of a plan does not necessarily approve every later action, especially when new information changes the recipient, amount or consequence.

Finally, specify expiry and evidence. Expiry can mean a deadline, completion of a case, revocation of a user's access, or consumption of a one-use authorization. Evidence should show the authorization decision and actual action outcome without copying credentials or unnecessary personal data into logs. A denied action is useful evidence too; repeated denials may reveal an integration mismatch or an attempted boundary crossing.

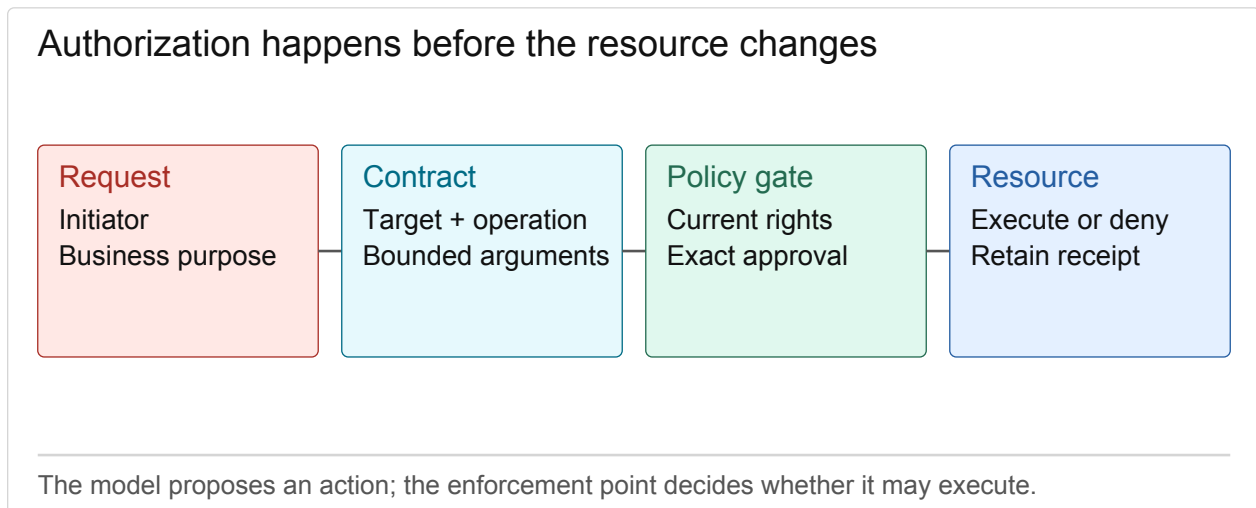


Figure 1. The action contract connects business purpose to an enforceable resource decision. Original Softlander conceptual diagram, 2026-09-16.

NIST's zero trust architecture separates authentication and authorization and rejects automatic trust based solely on network location or ownership. [2] Apply that distinction concretely: a correctly authenticated agent process can still request an action that exceeds the user's role, the task's purpose or the current resource state.

Put enforcement where the consequence occurs

There are three useful enforcement locations: before a tool call is issued, inside the tool adapter, and at the downstream resource. They have different visibility. A pre-call check can inspect the workflow request and proposed arguments. An adapter can normalize values, verify approvals and restrict the operation. The resource can apply authoritative object permissions and transaction constraints. Prefer complementary checks over assuming that one layer knows everything.

A pre-call classifier is useful for routing or flagging ambiguous requests, but it should not become the only permission authority for high-consequence actions. A model may misunderstand a request, and a policy prompt may not cover an unexpected representation. For a payment destination, for example, enforce a verified destination in the transaction system or adapter; do not rely only on a textual instruction to use approved suppliers.

Normalize before evaluating policy. The same destination may appear as an alias, redirect, encoded URL or indirect object reference. The policy evaluator and executor must agree on the resolved target. If the executor follows a redirect after approval, the effective destination may differ from the one reviewed. A safe design either resolves before authorization or rejects changes that invalidate the authorization.

For OAuth integrations, RFC 9700 recommends restricting access-token privileges to the use case and audience. [3] The architecture should therefore distinguish an authorization

token from a general credential that can be forwarded anywhere. MCP's security guidance explicitly identifies token passthrough as an anti-pattern and explains why validating the intended recipient matters. [4] Review each adapter's actual token handling, rather than inferring it from protocol branding.

An enforcement gate also needs a failure behavior. If the policy service is unavailable, an action that changes external state should not silently inherit permission from a cached conversational answer. Decide in advance whether to deny, queue for later evaluation or route to a human. Read-only operations may have a different fallback, but the distinction should be explicit and tested.

Control placement has cost. More checks can add latency, integration effort and support burden. Concentrate strict transactional controls on actions with material consequences while keeping low-risk reading paths usable. The point is not to surround every sentence with approvals. It is to make the few transitions that create obligations, disclose protected information or destroy state depend on reliable authority.

Design approvals that people can understand

A confirmation dialog is meaningful only if it shows the decision the person is about to make. "Allow agent to continue?" is usually too vague. A better approval presents the operation, target, important arguments, expected effect, expiry and whether the action can be reversed. It should distinguish a recommendation from the exact transaction that will be submitted.

Bind approval to those exact values. If the agent later changes the beneficiary, quantity, document set or external recipient, the previous approval should no longer authorize the altered action. An implementation can compare a normalized action record or a digest of the relevant fields. The important property is that the approved object and executed object cannot diverge unnoticed.

Separate the approver's competence from their availability. A manager may own spend but lack permission to disclose a personnel file. A security reviewer may understand access boundaries but not own a commercial commitment. Map approvals to the business consequence; use specialist consultation where needed without turning every participant into a universal approver.

A stalled approval must have a defined lifecycle. It may expire, be withdrawn, return for clarification or be reassigned under an existing organizational rule. Avoid keeping a privileged session open indefinitely while waiting. On resumption, recheck facts that may have changed: resource ownership, available budget, request state and the initiator's current rights.

High-volume work creates pressure to approve batches. Batch approval can be appropriate when the batch has a visible scope and bounded variation, such as sending an approved

template to a fixed recipient list. It is inappropriate when the agent can add new recipients or materially alter the message after approval. Display exceptions separately and allow the reviewer to remove individual actions without approving the rest by accident.

The authorization evidence should record who approved, what was approved and the policy version used. It should not depend on retaining a private chain of thought. An understandable action record, source facts, explicit decision and execution receipt provide a more practical basis for reviewing a business transaction.

Delegation needs an expiry path

Multi-agent designs can accidentally multiply authority. A coordinator delegates to a specialist, which invokes another tool with a broader service account. The parent task may have been narrow while the child's technical access is wide. Treat every delegation as a new authority boundary, even when both agents run inside the same application.

Require the delegated scope to be equal to or narrower than the parent's permitted scope. The child should receive only the resources, operations and lifetime needed for its work. If it needs more, it should return a request for a new decision rather than improvising a route through another credential. Keep the chain short enough that an operator can identify whose request caused an action.

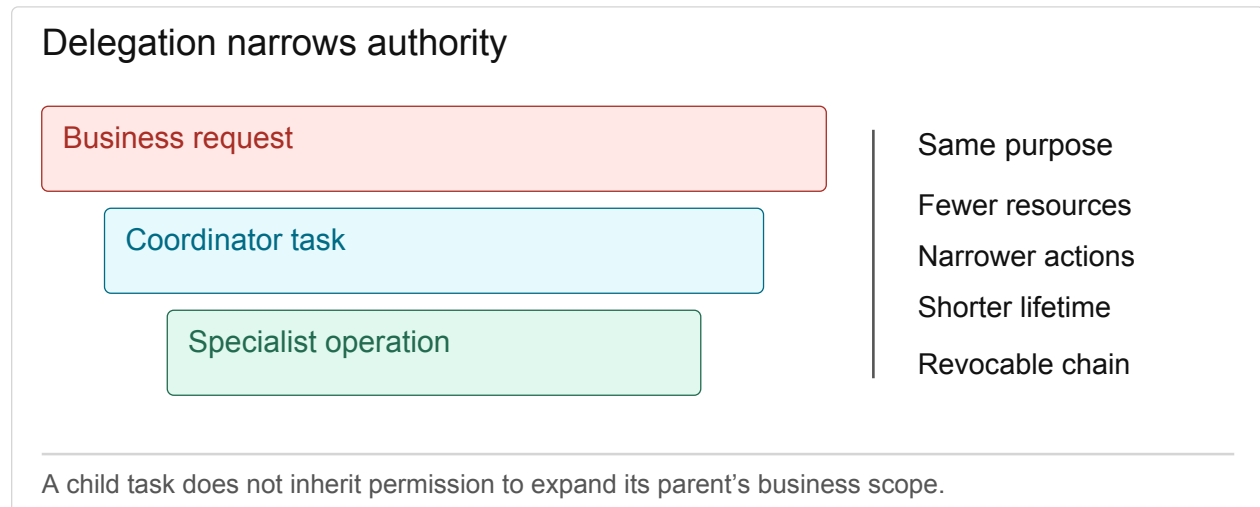


Figure 2. Each delegated task reduces scope and retains a traceable expiry and revocation path. Original Softlander conceptual diagram, 2026-09-16.

NIST SP 800-53 provides established control families for access control, audit, identification and authentication, and configuration management. [5] These are useful review anchors; the particular delegation contract here is an architectural recommendation, not a claim that applying this diagram implements the entire catalog.

Revocation should be tested at the boundaries where work can continue. Disabling a user in the directory does not prove that an already queued job, cached token or child task can no

longer act. Determine which actions are checked at execution time and which rely on earlier grants. For long-running jobs, define reauthorization points before consequential operations.

A cancellation request also needs semantics. "Stop" might prevent new actions while an in-flight transaction still completes. Communicate that distinction to operators and users. Preserve a receipt for completed work, identify uncertain outcomes and reconcile them against the authoritative system before retrying. Otherwise, an attempt to recover from interruption can create duplicate external commitments.

Where a platform cannot support fine-grained delegation, acknowledge the gap through a narrower product design. A specialist may return a draft for the coordinator to execute through one controlled adapter. Another option is to eliminate the child agent for that operation. Additional agent roles are useful only when their coordination benefits justify the extra authority and evidence boundaries.

Choose controls the integration can support

Before building a new authorization layer, inventory what the existing resource already enforces. A mature business application may provide object permissions, approval workflows and transaction limits that are stronger than an agent-specific wrapper. Reuse those controls where their semantics match the task, and add only the context that is missing. Duplicating a rule in several places creates a maintenance problem when the versions diverge.

Compare three integration options. A native constrained operation can keep enforcement close to the business record. A controlled adapter can narrow a broad API but becomes a security-critical component. A human execution step can preserve existing authority when the available API is too broad or poorly understood. The appropriate choice depends on the consequence, the platform's capabilities and the team's ability to test and operate the boundary.

For a purchased agent platform, ask the vendor to demonstrate the specific permission path. Can it distinguish users within a shared workspace? Can an administrator restrict operations and destinations? Does an approval bind the exact action? What happens to queued work after access is revoked? Request evidence from the relevant configuration, rather than relying on a general product statement about enterprise security.

A platform may expose logs without exposing the authorization decision that preceded the action. That gap can make investigation difficult. Determine whether the downstream resource provides the missing evidence and whether the two records can be connected. If the organization cannot reconstruct a consequential action, reduce its automated scope until the evidence path is adequate.

Also consider the operational cost of your chosen design. A custom gateway needs ownership, patching, policy change control and incident handling. A human gate needs informed reviewers and a usable queue. A native platform control needs configuration review and monitoring for vendor changes. Select the design whose obligations the organization can actually sustain; technical elegance does not substitute for an accountable operating model.

Worked example: supplier changes without silent authority expansion

Illustrative scenario. A purchasing team wants an agent to prepare supplier onboarding records. The agent can read an approved supplier questionnaire, compare it with the procurement policy and draft a record. The team is considering allowing it to activate the supplier and change payment details. No real company, deployment result or measured control effectiveness is represented in this example.

The first design workshop separates four actions: read the submitted material, create a draft record, activate the supplier and modify a payment destination. The procurement owner agrees that the first two can support the existing workflow. Activation requires completion of the organization's checks. Payment changes remain outside the agent's execution permissions and go through the established verification process.

The tool design mirrors that decision. Instead of a generic "update supplier" function, the adapter exposes a constrained draft operation. It accepts only approved fields and creates an unactivated record. It cannot set bank details or change activation state. The downstream credential is restricted to that operation where the system supports it. If the platform offers only a broad write credential, the adapter becomes a critical control and requires stronger testing and restricted access.

A malicious instruction inside the questionnaire asks the agent to use a different payment account. The expected behavior is not simply that the model refuses. The important test is that no available execution path can apply the payment change. The response may mention that the request is outside the workflow, but the security property is established by observing rejected arguments, unavailable operations and unchanged authoritative records.

The approval view for activation shows the supplier identity, evidence checklist, relevant exceptions and the exact state change. An authorized procurement approver can reject or return the record for correction. If the draft changes after approval, activation requires renewed approval. A system receipt links the activation event to the approved record version.

The operating team also tests a timeout after submitting activation. Before retrying, the adapter queries the authoritative record using a stable transaction reference. If it cannot determine the outcome, it escalates for reconciliation. It does not equate "no response" with

“no action.” This requirement belongs in the action contract because retry behavior can be as consequential as the initial permission check.

This design deliberately sacrifices some automation. The agent prepares work and reduces navigation effort, while sensitive authority remains in existing controls. A later decision to expand autonomy would require evidence about the new operation and an accountable owner; successful drafting alone would not justify that expansion.

Test the boundary, not just the happy path

Build a small test inventory around each action contract. Include a legitimate request, an unauthorized initiator, the wrong tenant or resource, an expired grant, changed arguments after approval, a revoked user, a delegated request that exceeds its parent, and a repeated transaction after an uncertain response. Define the expected resource state and evidence for each case before executing it.

Test the enforcement point directly as well as through natural-language prompts. If an adapter accepts a prohibited field when called directly, a well-behaved model has only hidden the defect. Conversely, an adapter that rejects an invalid action does not prove that the user experience explains the failure well. Assess security behavior and usable recovery separately.

Observe the gate and the consequence		
Test condition	Gate decision	Resource evidence
Valid request	Allow	Expected change
Wrong resource	Deny	No change
Stale approval	Reapprove	No change yet
Revoked parent	Deny	No child action

A reassuring response is not a substitute for inspecting the authoritative resource.

Figure 3. Negative tests compare authorization outcomes with the actual resource state. Original Softlander conceptual diagram, 2026-09-16.

Use fault injection carefully in an authorized environment. Simulate an unavailable approval service, delayed authorization response and ambiguous downstream receipt. Confirm whether the system denies, queues or escalates as designed. Retain enough environment and version information to interpret the result. A successful test against a mock does not establish that a production identity provider or resource behaves identically.

NIST's incident-response guidance places preparation, detection, response and recovery within ongoing risk management. [6] For this architecture, preparation means agreeing who can revoke authority, disable a connector and reconcile unfinished transactions. These are operational responsibilities, not just emergency buttons in a dashboard.

A practical review worksheet has six questions. What action creates the consequence? Where is the effective target resolved? Which identity is checked at execution? What exact facts does approval bind? What stops queued and delegated work after revocation? What record distinguishes a denied action, a completed action and an uncertain outcome? An unanswered question identifies a design task, not a reason to assign a reassuring score.

Start with the most consequential tool your agent can invoke. Draw the full authority path, write one action contract and run the negative tests against the actual boundary. If the only restriction you can identify is a sentence in the prompt, reduce the executable scope while the enforcement design is improved. The goal is a useful agent whose permission to act remains understandable after the conversation ends.

References

- [1] OWASP. [LLM06:2025 Excessive Agency](#). 2025 edition.
- [2] NIST. [SP 800-207: Zero Trust Architecture](#). August 2020.
- [3] IETF. [RFC 9700: Best Current Practice for OAuth 2.0 Security](#), section 2.3. January 2025.
- [4] Model Context Protocol. [Security Best Practices](#), token passthrough. Version 2025-11-25.
- [5] NIST. [SP 800-53 Revision 5: Security and Privacy Controls](#). September 2020, with release updates listed by NIST.
- [6] NIST. [SP 800-61 Revision 3: Incident Response Recommendations](#). April 2025.