



Practical AI Security and Governance

Five notes on putting AI agents to work with controls you can prove

AI assistants and agents are being connected to email, customer data and production systems faster than organizations decide who is accountable for them. These five notes follow one AI workflow from approval to operation: who decides, what the agent may reach, what it depends on, what stops it at runtime, and how to prove afterwards that the controls held. Each stands on its own; read together they are a sequence an MSP, MSSP or VAR can run with a client.

Telekomplex • telekomplex.net • October 2026

Contents

1. Before an AI Agent Acts, Decide Who Is Accountable
2. An AI Agent Needs Its Own Identity—and a Smaller Permission Set
3. A Clean Scan Is a Starting Point: Assessing AI Tools and MCP Servers
4. When a Document Tries to Give Orders: Layered Protection for AI Agents
5. Can You Prove the Controls Worked? Operating AI Security After Launch

Scenarios are illustrative recommendations, not reports of actual deployments. Vendor references are sourced examples; capabilities and documentation change, and source links were reviewed on 1 October 2026.

Before an AI Agent Acts, Decide Who Is Accountable

An AI system's capabilities can grow faster than the decisions about its use. Approval for one task should not silently become approval for every future capability.

A sales team connects an AI assistant to its customer database. The pilot works well: it summarizes conversations and suggests follow-ups. Someone then enables email sending. Another person adds access to pricing information. Within weeks, a writing assistant has become a system that can make commitments to customers.

Who approved that change in responsibility?

This hypothetical situation illustrates a governance problem: an AI system's capabilities can expand faster than the organization's decisions about its use. Approval for one task should not silently become approval for every future capability.

The first control is a clear description of the intended use. Document the business objective, the people affected, the information involved, and the actions the system may perform. Distinguish between producing a recommendation and executing a transaction. A draft discount proposal and an automatically issued discount carry different consequences.

Assign responsibilities that people can actually exercise. A business owner defines acceptable outcomes and accepts residual business risk. A technical owner maintains the integration. Security and privacy specialists assess relevant exposure. An operational owner can suspend the service and coordinate recovery. One person may hold several roles, but an ownership label is useful only if it comes with authority, time, and an escalation path.

The NIST AI Risk Management Framework offers a voluntary structure for managing AI risk through Govern, Map, Measure, and Manage. These functions support continuing risk management; adopting the framework does not certify that a particular system is safe. [NIST AI RMF \[1\]](#)

For implementation, I suggest a short deployment record that follows the system through its lifecycle. It should identify the approved purpose, model and tool configuration, accessible data, prohibited actions, necessary approvals, evaluation results, known limitations, and review date. Record who can approve a material change, including a new data source or broader tool permissions.

Consider the sales assistant again. Its initial approval could permit reading assigned customer records and preparing email drafts. Sending an external message would require a separate decision, supported by tests of recipient selection, factual accuracy, confidential information handling, and escalation when the assistant lacks evidence.

Governance also extends beyond cybersecurity. A system can protect credentials and still generate misleading advice, treat customers inconsistently, or create an inaccessible service. Review the foreseeable effects on people, the reliability needed for the task, and how someone can question or correct an outcome.

Human oversight should have a defined purpose. Give reviewers the proposed action, supporting information, uncertainty, and likely consequences. If a reviewer sees only an approval button, the organization has created a checkpoint without providing the means to make an informed decision.

A practical starting point is one consequential workflow. Map its decisions, assign owners, define the operating limits, and specify what evidence would justify expansion. Connect that record to change management so that adding a tool or replacing a model triggers the appropriate review.

The test of governance is whether an organization can explain why an AI system is allowed to act, within which boundaries, and who can intervene when those boundaries no longer fit.

Before expanding your next pilot, ask: which capability would require a new approval, and who has the authority to give it?

Sources

[1] <https://www.nist.gov/itl/ai-risk-management-framework>

An AI Agent Needs Its Own Identity—and a Smaller Permission Set

An agent borrowing a manager's credentials has more authority than its task requires before the model makes any mistake. Give it its own identity and a narrower permission set.

Imagine a customer-support agent using a manager's credentials. It can read tickets, issue refunds, export customer records, and change account settings because the manager can do all four. The agent's actual assignment is much narrower: summarize a ticket and prepare a response.

The problem exists before the model makes a mistake. The integration has granted more authority than the task requires.

OWASP describes excessive agency as a risk arising when an AI application has too much functionality, permission, or autonomy. Reducing that exposure means examining what the application can execute, not simply improving the instructions given to the model. [OWASP: Excessive Agency \[1\]](#)

Start by separating three concepts. Identity establishes which software actor is making a request. Authorization determines whether that request is permitted. Accountability identifies the people and teams responsible for operating the system. Mapping an agent to an owner does not mean the owner personally approved each action.

A useful access design gives the agent a distinct, managed identity and grants permissions for its specific workflow. Where it acts on behalf of a user, preserve that delegation context without copying the user's entire permission set. Effective access should respect both the application's limits and the user's permitted scope.

For our support example, I would allow the agent to read the assigned ticket and create an internal draft. Customer-data exports and account changes would be unavailable. A refund workflow would use a separate, narrowly defined tool, with server-side checks on the ticket, customer, amount, and approval.

This distinction matters because allowing a tool name is not enough. A permitted “read customer” operation could still request the wrong customer's record. Enforce resource and tenant boundaries in the service that owns the data, even if a gateway also performs checks.

Credentials should be managed outside prompts and ordinary configuration files, using an appropriate secret manager or workload identity mechanism. Prefer scoped, short-lived credentials where the integration supports them. Short lifetimes reduce exposure; they do not automatically provide immediate revocation.

For HTTP-based MCP authorization, validate tokens for the intended resource. The MCP security guidance explicitly rejects token passthrough: accepting a token intended for another service and forwarding it downstream undermines authorization boundaries. Downstream access needs its own correctly designed authorization arrangement. [MCP security guidance](#) [2]

Product roles also deserve precision. Cisco describes Duo as supporting agent identity and lifecycle management, and Secure Access as enforcing fine-grained access policies through its MCP gateway architecture. Those are complementary functions; identity registration alone does not establish complete enforcement. [Cisco solution brief](#) [3]

Finally, test removal of access. Disable the agent, change its owner's role, and exercise old sessions and cached credentials in a controlled environment. Measure when access actually stops, including queued jobs and downstream connections. A directory update is an event; successful revocation is an observed outcome.

A simple access record can capture the agent, owner, permitted action, resource scope, conditions, credential lifetime, approval requirement, and revocation mechanism.

The question to ask is concrete: if this agent misunderstood its task today, what would the infrastructure still prevent it from doing?

Sources

[1] <https://genai.owasp.org/llmrisk/llm062025-excessive-agency/>

[2] https://github.com/modelcontextprotocol/modelcontextprotocol/blob/main/docs/docs/2026-07-28/tutorials/security/security_best_practices.mdx

[3] <https://www.cisco.com/c/en/us/solutions/collateral/artificial-intelligence/security/zero-trust-agentic-ai-sb.html>

A Clean Scan Is a Starting Point: Assessing AI Tools and MCP Servers

Using the Model Context Protocol does not establish that a server, its maintainer or the systems behind it can be trusted. Approve an exact version for an exact task, on evidence.

A team discovers an MCP server that promises convenient access to business data. Its demonstration looks convincing, the tool descriptions sound reasonable, and a scanner reports no findings. The team approves the connection.

Two weeks later, the server changes.

Which version did the organization approve, and which behavior did it actually evaluate?

The Model Context Protocol provides a way to connect AI applications with capabilities such as tools and data resources. Using that protocol does not, by itself, establish the trustworthiness of the server, its maintainer, or the systems behind it.

Supply-chain assessment should therefore begin with provenance and purpose. Identify who maintains a component, where it runs, how it is updated, which dependencies it includes, and what data leaves the organization. For a hosted service, establish what evidence the provider can supply and which changes customers can observe or control.

My suggested admission process separates description, implementation, and behavior. A tool description says what the tool claims to do. Code and configuration can reveal how it works when available for review. Controlled execution can show what happens under selected conditions. None of these views completely replaces the others.

Cisco's open-source MCP Scanner illustrates how several assessment methods can contribute. Its documented options include YARA rules, LLM-based analysis, the Cisco AI Defense inspection API, and modes for examining exposed MCP content, available source code, and dependencies. Coverage depends on the selected mode, configured analyzers, and accessible material. A metadata-only scan cannot establish everything the server's code might do. [Cisco MCP Scanner \[1\]](#)

Treat findings as evidence for a decision. An absence of findings means the configured checks did not identify a problem in the material examined. It does not establish that every future response or tool execution will be safe.

For a hypothetical customer-record connector, I would first document the permitted use: retrieve a limited set of fields for an assigned account. Then I would test whether requests for another account fail,

whether the connector attempts unexpected network connections, and whether an update changes its permissions or returned content.

Run unfamiliar components in an isolated assessment environment with synthetic data and restricted credentials. This applies to the assessment process itself: a scanner mode that launches a local server executes that server, whereas an offline metadata scan has a different exposure.

The approval record should identify the artifact or service version where available, configuration, tool inventory, assessment date, findings, accepted limitations, reviewer, and reassessment triggers. For self-managed software, pin reviewed versions and track updates. For hosted services that cannot be pinned, use contractual change information, monitoring, and repeat evaluations to manage that uncertainty.

Extend the same discipline to models, retrieval sources, agent instructions, and other dependencies. A model replacement may change workflow behavior even when its API stays compatible. A document repository can introduce new content without changing the model at all.

For an initial rollout, create a small approved catalog with named maintainers and explicit reasons for admission. Make removal and rollback part of the catalog process, so that discovering a problem produces an actionable response.

The useful question is broader than whether a scanner passed: what evidence supports using this exact capability, for this task, with this level of access—and what change would invalidate that decision?

Sources

[1] <https://github.com/cisco-ai-defense/mcp-scanner>

When a Document Tries to Give Orders: Layered Protection for AI Agents

Indirect prompt injection arrives inside the content an agent is processing. The objective is to keep untrusted content from ever acquiring operational authority.

An employee asks an AI assistant to summarize a supplier proposal. Inside the document, an attacker has placed instructions telling the assistant to retrieve internal pricing and send it to an external destination.

The employee requested a summary. The document is trying to create a new assignment.

This is a hypothetical example of indirect prompt injection: instructions arrive through content the AI system is processing rather than through the authorized user's request. OWASP identifies this risk and recommends combining controls such as privilege restrictions, output validation, filtering, and oversight. No single filter should be treated as a complete solution. [OWASP: Prompt Injection \[1\]](#)

The practical objective is to prevent untrusted content from acquiring operational authority. Marking a document as untrusted can help the model interpret it, but an instruction to “ignore malicious content” does not replace controls over what the application can execute.

For the supplier workflow, I would make summarization possible without giving the assistant access to internal pricing or an external sending tool. A compromised summary process would then have fewer ways to cause harm. If another workflow requires those capabilities, give it separate permissions and explicit approval conditions.

Runtime protection should follow the actual path of the data and actions. Inspect incoming material before it enters the model context when that boundary needs protection. Validate a proposed tool call before execution. Check returned tool data before passing it to the model where the application requires that safeguard. Review outgoing content before delivery to its destination.

Placement determines what a control can accomplish. An inspection performed after a model has received sensitive information cannot undo that disclosure. A gateway cannot inspect a direct connection that bypasses it. Draw the routes, including local tools, background jobs, and external APIs, before claiming coverage.

Consider an authorized operation that creates a supplier record. The application should validate field types, permitted values, tenant ownership, and the caller's authority. For an external message, inspect the actual recipient and attachment selection. These checks constrain execution even when the proposed action sounds reasonable in natural language.

Content inspection adds another source of evidence. It can identify suspected injection, sensitive information, or policy violations, but requires evaluation for both missed threats and unnecessary blocking. Decide how the workflow behaves if an inspection service is unavailable. For a consequential action, pausing for review may be preferable to proceeding without the required check.

Human approval should bind to a concrete action. Show the recipient, content, data involved, and expected effect. If those details change after approval, require a new decision where the change affects risk. A vague approval to “continue” offers little protection against a materially different operation.

Cisco positions AI Defense as a runtime protection component; DefenseClaw describes itself as an enforcement and evidence layer with scanning and inspection integrations. These tools need deployment-specific coverage and policy validation. Their presence does not establish universal blocking of every attack. [Cisco architecture](#) [2], [DefenseClaw](#) [3]

A useful test is to place a harmless simulated malicious instruction in a test document and observe the complete workflow. Success means the intended task remains useful while unauthorized access or execution is prevented and recorded.

The boundary that matters is where a model's suggestion becomes a real-world action.

Sources

[1] <https://genai.owasp.org/llmrisk/llm01-prompt-injection/>

[2] <https://www.cisco.com/c/en/us/solutions/collateral/artificial-intelligence/security/zero-trust-agentic-ai-sb.html>

[3] <https://github.com/cisco-ai-defense/defenseclaw>

Can You Prove the Controls Worked? Operating AI Security After Launch

A blocked request on a dashboard is not proof of containment. Operating AI security means evidence that connects decisions to their effects.

An AI agent attempts to export customer information. A dashboard shows a blocked request, and the team closes the alert.

Before declaring the incident contained, several questions remain. Was this the only attempt? Did an earlier request succeed? Could a background job still run? Which system recorded the final outcome?

Operating AI security requires evidence that connects decisions to effects.

NIST's incident-response guidance places preparation, detection, response, and recovery within continuing cybersecurity risk management. AI systems should join that operational process, with additional attention to their models, tools, data sources, and delegated actions. [NIST SP 800-61 Rev. 3](#) [1]

For an agent workflow, I suggest an event record that identifies the agent, relevant user or service context, accountable owner, requested operation, target resource, policy version, decision, and execution result. Use correlation identifiers to connect the application, gateway, tool server, and downstream service.

Keep the distinction between request and result visible. "Allowed" means an enforcement point permitted an operation. It does not establish that the operation completed. "Blocked" describes a particular intervention; it does not establish that every route to the same outcome was blocked.

Collect enough information to reconstruct material events without turning the audit system into a new repository of secrets. Redact or exclude credentials, restrict access to sensitive records, and define retention according to operational needs and applicable obligations. Record actions and observable outcomes; do not rely on a model-generated explanation as proof of what happened.

An illustrative incident exercise can make these requirements concrete. A test agent tries to export synthetic customer data to an unapproved destination. The team must identify the attempt, confirm whether any data left, stop the relevant workflow, and preserve evidence. Then it must demonstrate that retries, queued work, and alternative connections cannot continue the operation.

Disabling an agent identity is one containment step. Depending on the architecture, responders may also need to revoke credentials, terminate processes, cancel jobs, restrict network access, or disable a connector. The runbook should state which team controls each mechanism and how to verify its effect.

Recovery deserves equal attention. Review changes the agent already made, repair affected records when appropriate, rotate exposed secrets, and correct the underlying weakness. Restart under a controlled configuration after the relevant tests pass. Record the remaining uncertainty rather than turning an incomplete investigation into a claim of complete safety.

Build a repeatable evaluation set that includes legitimate work as well as misuse. Otherwise, a system that blocks every request could appear secure while being operationally useless. Re-run relevant tests after material changes to the model, instructions, tools, permissions, or inspection policy.

Useful measurements include the proportion of consequential actions with complete audit records, tested misuse attempts that achieved a prohibited outcome, legitimate tasks incorrectly blocked, and observed time to contain a controlled incident. Report the test conditions and denominator. A result from a small test set is evidence about that test, not a universal protection rate.

DefenseClaw's documented emphasis on durable audit evidence is one example of tooling that can support this process. Organizations still need to connect that evidence with identity records and downstream outcomes. [DefenseClaw repository](#) [2]

The monthly review should produce decisions: narrow an agent's scope, repair an integration, improve an alert, accept a documented residual risk, or retire a capability.

The operational question is simple: if an agent acted outside its assignment this afternoon, could your team establish what happened, stop further harm, and justify a safe return to service?

Sources

[1] <https://csrc.nist.gov/pubs/sp/800/61/r3/final>

[2] <https://github.com/cisco-ai-defense/defenseclaw>

Delivered under your brand

Telekomplex gives MSPs, MSSPs and VARs certified engineering capacity to govern, secure and operate AI for their clients — from employee AI use and agent identity to runtime protection and AI security operations — delivered white-label, inside your delivery model.

telekomplex.net/ai-security • contact@telekomplex.net