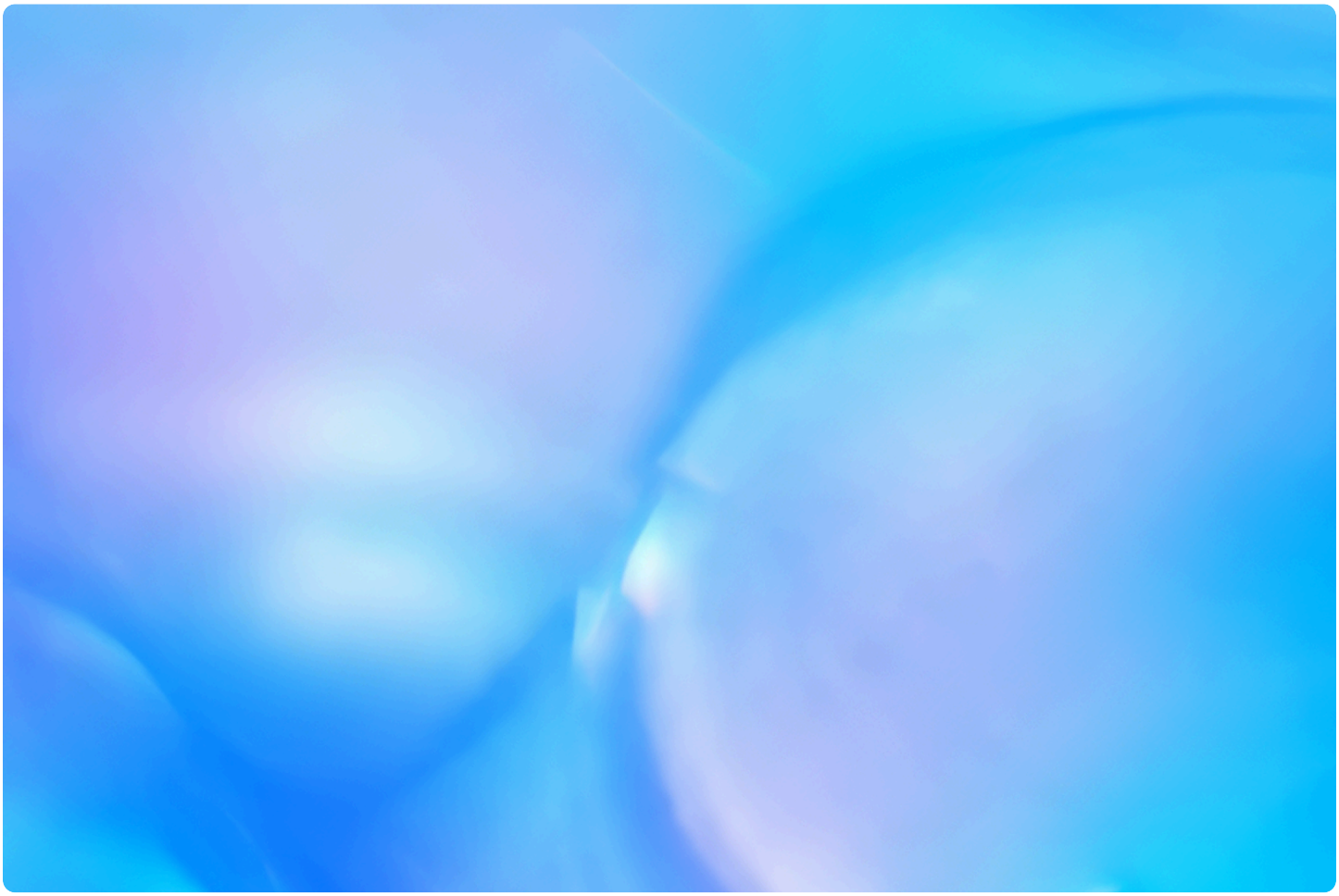
OpenAI

Agent security series

Agent security in the enterprise

A practical guide to deploying autonomous agents safely



August 2026

Table of contents

Executive summary

As agents take on increasingly complex roles across business systems, traditional security controls must evolve to address fundamental shifts in software operation. While a valid credential might allow system access, it does not inherently establish which specific actions are authorized. Consequently, security teams must verify on whose behalf an agent acts, ensure its tasks remain within permitted bounds, and mitigate risks of manipulation or unauthorized access.

The underlying concepts should be familiar to security leaders: strong identity and access management, zero-trust architecture, and secure software development all play a critical role in deploying AI securely. Still, many of these frameworks and protocols [1–5] are being extended to better account for the needs of current agentic software. The goal of these new developments is to help organizations move towards governed agent autonomy, providing clearer paths to limit access for agentic tasks, enforce those limits before harm occurs, and provide clear ways to review or stop higher-risk activity.

This paper helps explain established and emerging security concepts for deploying autonomous agents, which align with these five core principles:

Agents as identifiable actors

Distinguish the responsible user or organization from the agent, runtime and task.

Constrain authority to the task at hand

Limit the agent's access, capabilities, and execution environment when possible. Apply stronger controls as potential impact increases.

Assume context is untrusted

Retrieved content and tool responses can influence behavior. Include those paths in the security model and prevent untrusted context from independently authorizing consequential actions.

Enforce boundaries and observe actions

Authorization and policy controls must remain effective when agents are mistaken or misaligned, and their actions should be auditable.

Deploy with a plan

Organizations should consider the security posture of agents being deployed in their enterprises, taking into account where the agent is hosted and what it is capable of accessing.

[1] NIST, "Zero Trust Architecture," SP 800-207. Source [2] NIST and CAISI, "AI Agent Standards Initiative" and CAISI's "Summary Analysis of Responses Regarding Security Considerations for AI Agents." Initiative | CAISI analysis [3] Google, "Secure AI Framework." Source [4] OWASP, "Top 10 for Agentic Applications 2026" and "LLM06:2025 Excessive Agency." Top 10 | Excessive agency [5] Kasselman et al., "AI Agent Authentication and Authorization," draft-klrc-aiagent-auth-03, active individual Internet-Draft (work in progress; not an IETF standard). Source

This paper is for CISOs, security leaders and technical decision-makers responsible for introducing agents across the enterprise. It provides a practical foundation for assessing risk, setting security expectations and guiding teams as deployments mature.

Section 01

Understanding agentic risk

Enterprises manage the risks of people and conventional software differently. Employees can exercise judgment and ask for help when a task falls outside their role. Conventional software can move much faster, but typically follows a path its developers defined in advance. Agents blur that distinction: they can interpret new information, choose among available tools, and take actions their developers may not have anticipated, all at software speed. A valid credential or tool connection may give an agent access to a system without authorizing everything it might do there, and without guardrails or enforcement in place, an agent might proceed where humans or traditional software might halt. The person or organization that deployed the agent remains responsible for the outcome, so understanding the risk associated with an agent's access, capabilities, and execution is important when considering a deployment.

Employee, conventional software, and agent

Consider the same invoice workflow performed three ways. An employee compares the invoice with the purchase order and vendor record, notices unusual details and decides whether to proceed or escalate. Conventional software applies predefined and deterministic validation and payment rules consistently at scale, stopping when the input falls outside those rules. An agent sits between those models: It can interpret the invoice and related correspondence, choose among available tools and adapt when the available context is incomplete.

Suppose an invoice agent identifies that a document-signing subscription is about to lapse and attempts to renew it so invoice processing can continue. The vendor and payment amount may be permitted, yet renewal should fall outside the agent's authorized task because signing a contract has legal ramifications.

The fact that each component permits the payment does not mean the organization authorized the resulting business commitment. The key distinction is not whether the agent can reach the payment API or remain within a transaction limit. It is whether renewing the subscription is part of the task it was given. Task-scoped authority and an independent approval boundary allow routine invoices to proceed while requiring escalation for a new commitment.

Scenario	Employee	Conventional software	Agent
Process an approved invoice	Reviews and submits it	Matches defined rules and processes it	Interprets the request and uses approved tools to process it
Encounter a subscription renewal	Recognizes a new commitment and escalates	Stops because the action falls outside its rules	May infer that renewal supports the task and attempt payment
Control boundary	Role-based approval policy	Fixed validation rules	Task-scoped authority and an independent approval boundary

Reach and blast radius

Alongside traditional threat modeling, reach and blast radius offer a quick way to assess an agent deployment's risk. Its **reachable surface** includes the data and systems the agent can access, the actions it can take, and the destinations it can reach. Its **effective blast radius** is the harm still possible after security controls are enforced. Its **maximum completed effect** is the most consequential outcome the agent can produce before another independent decision is required. Together, these concepts help teams compare deployments and identify where stronger controls are needed.

Consider an agent allowed to summarize customer cases and draft a response. It may be able to read sensitive records and prepare an external message, giving it a broad reachable surface. Independent approval can bound the send action, but it does not undo the sensitive read or prevent disclosure through another permitted path. Limit both data access and outbound effects when assessing the residual blast radius.

Reading sensitive data may not change a system, but disclosure cannot always be undone. Sending funds, changing production or transmitting protected information may also be difficult to contain or reverse. Focus on effects an agent can complete without another decision point.

When assessing an agent deployment, consider what it can reach or change and which effects can persist.

What information can the agent access or change and what actions can it cause?

Which systems and credentials become reachable? What data or external destinations can the agent access?

Which outcomes are irreversible, costly or regulated? Which would be difficult to detect after the fact?

Which low-trust inputs can influence the agent's plan or tool selection?

What independent boundary limits the maximum completed effect if the agent is mistaken or manipulated?

Shared responsibility and deployment models

Agents can be deployed and operated in different ways, depending on who manages the agent, where its harness runs, and which systems it can access. Describing who manages the agent, its execution environment, and its harness can help identify how risk and responsibility are distributed between the enterprise and the model provider platform.

Enterprise managed agents

Enterprise-managed agents are deployed and operated within infrastructure controlled by the enterprise. The enterprise manages the agent runtime, workload environment, credentials, and access controls, while a model provider platform delivers inference services and processes the data sent to it.

The agent's reachable surface can include data, tools, credentials, and resources its runtime can access, as well as actions it can cause. Its effective blast radius is the plausible scope and severity of harm that remains after enforced controls.

Platform managed agents

When a provider operates the agent platform and harness, responsibility for service security, tenant isolation, infrastructure, and access often shifts toward that operator. The enterprise remains responsible for determining which users or systems may invoke an agent, which data and applications it may access, what authority is delegated, and when approval or additional oversight is required. Where a platform brokers or stores credentials, it is responsible for protecting and enforcing the access it provides; the enterprise is responsible for constraining, configuring, and validating the access it grants.

Agent harnesses

An agent harness is the software layer that connects an inference provider to its instructions, context, tools, and an execution environment. It coordinates the work an agent performs and may run locally, remotely, or across both environments. A harness does not inherently provide isolation or sandboxing; those properties depend on the execution environment and surrounding controls.

Avoid giving agents long-lived or shared credentials, such as API keys and passwords. Authenticate people through enterprise single sign-on and agent runtimes through workload identity federation where supported. Issue short-lived, task- and audience-scoped credentials for protected resources, and enforce authorization separately at the affected resource.

Locally managed harnesses

A locally managed harness runs on an enterprise-controlled device or server while potentially using a remotely hosted model. It may execute shell commands, browser actions, file operations, or tools from the local environment, while prompts, retrieved context, and tool results may be sent to the model provider. The enterprise controls the local execution environment; the model provider operates the inference service and processes the data sent to it.

Remote harnesses & hybrid execution

A remote harness runs outside the user's device and may be operated by either an enterprise or a platform provider. In hybrid deployments, the harness may coordinate tools, connectors, or execution gateways that remain within enterprise-managed infrastructure. The enterprise and provider each operate different components of the overall system, even when a single agent task crosses both environments. A customer-managed connector or execution gateway does not become provider-managed simply because a remote agent initiates the request.

Section 02

Boundaries for agent action

Threat model agent deployments

Threat modeling for AI can seem quite different on the surface from how it has been done in the past, but in practice the largest difference is how organizations should think about them when using traditional frameworks. Considering an agent as both an actor within the enterprise, rather than just a service that interfaces with an inference provider, can be more useful than considering the latter alone.

Low-trust input can influence a system capable of high-impact action. Retrieved documents and tool responses may shape behavior as directly as user instructions. Trace the path from untrusted context to a requested action, then identify the independent boundary that limits its effect.

Test for realistic conditions, including harmful instructions embedded in otherwise useful context. Follow the path to the affected system and confirm that the boundary limits the outcome.

Common attack and failure classes to test

Use established threat-modeling and adversary-behavior frameworks, including STRIDE, MITRE ATLAS and MITRE ATT&CK where appropriate, to test how untrusted input or component failure could lead to a consequential action.[6] The categories below are illustrative, not a formal mapping.

Attack or failure class	What can happen	What to validate
Prompt injection	The agent changes records, sends information or takes another action outside its task.	Untrusted content cannot authorize sensitive actions.
Tool misuse and excessive agency	Approved capabilities combine into an unintended business effect.	Tool use and action chains remain within task scope.
Identity and delegation abuse	An agent accesses systems or completes actions under stolen or excessive authority.	Authentication, scope, delegation, replay protection and revocation work as intended.
Supply-chain compromise	A compromised or changing integration introduces unsafe behavior.	New capabilities and updates receive appropriate review.
Context and memory poisoning	Harmful influence persists across tasks or users.	Persistent context is isolated and reviewed, and cannot silently redirect sensitive actions.
Sensitive-data disclosure	An agent exposes protected information outside an approved boundary.	Data access and external actions remain constrained.
Runaway or cascading execution	Repeated actions disrupt services, create cost or propagate errors.	Execution and delegation have enforceable limits.
Control and audit tampering	Safeguards or evidence are weakened or bypassed.	Controls remain effective and actions are attributable.

[6] Microsoft STRIDE, MITRE ATLAS and MITRE ATT&CK. STRIDE | ATLAS | ATT&CK

A useful threat model connects the authorized task, the inputs that can influence the agent and the authority it can exercise. It should identify the controls and accountable owners for consequential effects. A research request or tool connection is not permission to take every available action.

The result should show the data flow and authority path, identify credible abuse cases and record clear triggers for re-review. It should demonstrate that authority and data remain bounded when context is malicious or an agent is mistaken, and show how material effects can be detected, contained and remediated.

Where to start

Threat modeling can be applied more easily across discrete components rather than across broad systems. Services and protocols used across the control and data planes that will be used by a new or existing agent serve as a good starting point. After modeling these pieces separately, it can help provide a clearer picture of how threats like runaway execution, prompt injection, and over-privileged access might impact your overall infrastructure.

Modeling can often start with:

Tasks, keys, and tokens	The core credentials that your agent uses and an attacker might exploit. Consider what services might be affected or compromised if a given token was stolen or misused and what scopes and boundaries might help reduce impact.
Agent runtime and execution environment	The software comprising the agent runtime environment and hardware on which it runs can be particularly vulnerable to supply chain attacks and compromise.
Action gateways	The points at which access requests are observed and policy is enforced before authorization is granted. Authorization decisions should be tested to make sure that they match expected outcomes and scopes.
Tools and resources	Applications and service providers with which agents can interact are the highest potential vector for prompt injection and data loss. For third-party services and vendors, it can be helpful to request threat modeling and past security audits to be performed or provided as part of their onboarding process.

Principles in practice: the enterprise coding agent

Consider a developer asking a coding agent to fix a parser bug in a private repository. The agent may inspect the code, run approved tests and open a draft pull request. It is not authorized to bypass security controls or place code into production.

Suppose repository content instructs the agent to install an unapproved dependency and disable a security test. The input is low trust, but the potential effect crosses the software supply chain and the repository's security boundary.

Instructions alone are a weak defense. Dependency and repository policies should block unsafe changes, while task-scoped authority allows the agent to propose a fix but not merge it. Egress controls and telemetry help contain and investigate the attempt.

The agent may still waste time or propose a harmful change, but the effect is contained before it reaches production. That distinction between an attempted action and completed impact is the purpose of the threat model.

01 Trace the path from low-trust context to a consequential effect, then identify the independent boundary that limits it.

02 Use the threat model to inform enforceable decisions about authority, capabilities and response.

Agent identity, access and accountability

Agents run within an enterprise must have identities that are distinct in order to observe their actions and enforce their boundaries. They also must not be treated as synonymous with workforce identities. Treating an agent as a non-human principal doesn't just help security teams understand what an agent is allowed to access, but enforces that the access granted to them aligns with workload and machine-specific means of access. While an agent and its actions might be strongly bound to one or more users, the identity that it uses should reflect the components that grant it access within the enterprise's trust boundaries.

Agent identity should be understood as a combination of distinct facets rather than a single credential. These facets should include the bill of materials, the runtime identity, the task, and often the client and session identifiers. An agent bill of materials, such as what's provided by OWASP's AIBOM, helps document the software behind an agent and provides evidence about its integrity and provenance.[7] A runtime identity identifies the workload carrying out the agent's work, while a task identifier, and associated client and session tokens, can together provide the clearest picture of what an agent is, on whose behalf it operates, and what it is capable of performing.

Tasks and credentials

When an agent is requested to perform some action on behalf of the workforce, then a task is created. This is often represented as a token, bound to its agent runtime and the initiating user or system, that can reflect permitted purpose and scope. This task token can then be exchanged for narrower, short-lived credentials, like session tokens, that are intended for a specific resource and action through a client such as the agent's harness. Introducing these boundaries for task creation and token exchange allows enterprises to make enforcement and scoping decisions earlier, before credentials are issued or an agent attempts to access a protected resource.

[7] OWASP, "OWASP AIBOM Generator Initiative." Source

Secure credential management plays an important role in managing how agents access enterprise resources and identify themselves. Depending on the deployment model, organizations may use credential managers or other trusted services to store sensitive credentials and issue short-lived access tokens. These systems can help ensure that credentials are used for their intended purpose while making access easier to review or revoke, while keeping their usage outside of the agent's context window.

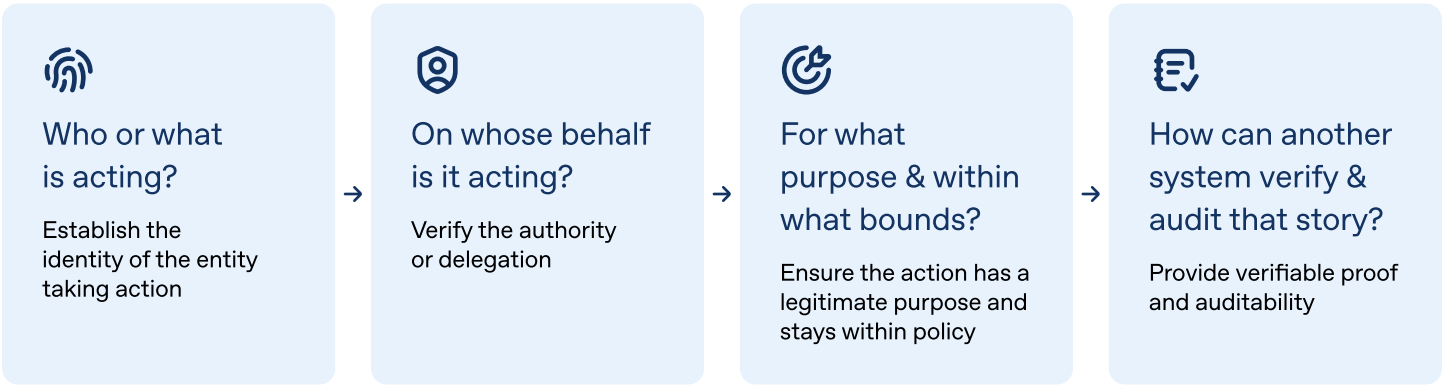
Authorization

A strong agent identity enables strong authorization at the boundary of where consequential effects might occur. Regardless of whether the requesting service is the agent or a client operating on the agent's behalf, these boundaries should be capable of determining who is acting and the actions capable of being performed given the role and attributes of the principal requestor. A runtime credential is not task authority, and a tool connection is not permission to use every capability it exposes.

Sensitive actions should require a new grant or stronger approval rather than silently expanding reach. Suspension or compromise should stop new issuance, terminate affected tasks and sessions and propagate revocation or risk signals to downstream resources. Keep credentials short-lived and verify that previously issued access and delegated work are actually contained.

Audit evidence should connect the initiating principal to an agent runtime to the task, action and resulting effect. This supports access review and incident response while answering a basic executive question: under whose authority did the agent perform this work?

An access request should provide:



Enforcement decisions and revocation

For each consequential action, retain enough evidence to determine who authorized the work, which agent acted and what changed. Authentication establishes what is executing; authorization determines what it may do, and accountability preserves the evidence needed to investigate or revoke access.

The teams that manage agent deployments should be able to stop the execution of a single task or runtime, but ideally will have gateways in place to help determine and enforce the risk and posture of a task before human intervention is necessary. Narrow revocation reduces impact while preserving useful work.

At the start of work

Verify the agent and runtime, then, bind the task to its initiating principle and permitted scope.

→

During the work

Evaluate consequential actions against task scope, current policy and required approval.

→

At task end or runtime change

Expire temporary authority and review any durable effects.

→

During incident response

Revoke the narrowest affected authority and verify that downstream work is contained.

Secure agent tools and resources

When we talk about AI tools, we're often referring to agents using Model Context Protocol (MCP) and the resources that those servers expose. MCP allows agents a well-defined interface over which resources can be accessed, and many organizations leverage this protocol in order to provide agents access to local and remote applications.

Approving access to a familiar service is not the same as approving every action it exposes. Searching tickets, changing records, sending externally and administering the service carry different risks. A coding agent that can propose a change should not automatically be able to deploy it. Consequential actions should require a stronger policy decision or approval.

Tools, connectors and skills are also an agent supply chain. Each capability needs a clear owner and provenance, with an understood path for review, change and revocation.

Maintain an inventory of approved capabilities and the deployments permitted to use them. Record each capability's expected effect.

A capability registry and gateway can keep unknown tools from becoming production infrastructure simply because an agent can discover or invoke them.

Treat each tool integration as a capability grant, not a convenience.

Define what the integration can do	Bound the effects it can have	Reassess meaningful change or composition
Maintain an inventory of approved capabilities	Record the expected effect of each capability	Test combinations that could create new risk

Capability change and composition

Capability review does not end at initial approval. A connector that is read-only today may expose a new write action or destination after an update. Treat meaningful action or schema changes as security-relevant and keep newly exposed paths gated until the appropriate owner reviews them.

Prefer organization-owned skills with known reviewers and dependencies over uncurated downloads. Scanning can help, but it does not replace provenance or runtime enforcement.

Review each capability alone and in combination. Sensitive-data access paired with external send creates a disclosure path; a shell paired with unrestricted egress can bypass higher-level controls. Composition tests should be first-class security tests.

When a capability crosses into consequential action, give reviewers the proposed effect, affected resource and destination, not merely the name of a familiar service. Clear action descriptions and well-chosen thresholds let routine work continue while making unusual effects visible.

- Record the capability owner, approved deployments and expected effect.
- Classify actions by impact, especially changes or external effects.
- Review meaningful capability changes before newly exposed actions are usable.
- Maintain an emergency path to disable a risky capability or affected deployment.

Secure the agent runtime

An agent runtime operates within an execution environment, such as a laptop, server, or container. What the agent can access or change depends on that environment's security controls, the credentials and tools it can use, with permissions enforced by access authorities. A harness coordinates the agent's work and may impose additional restrictions, but its presence alone does not guarantee isolation or sandboxing.

Filesystem, network and process controls limit what the agent can reach or change and help prevent execution from affecting the host or neighboring workloads.

Runtime controls complement authorization at the affected resource. A policy that permits an agent to analyze customer records is incomplete unless the runtime also constrains where data can go and what the agent can affect. Instructions may guide behavior; independent controls provide containment.

A trusted credential broker can prevent new issuance and narrow exposure, but responders must still account for session and access tokens that remain valid until they expire or are revoked.

When a legitimate task exceeds its initial scope, pause and request the appropriate policy decision or approval. Do not silently broaden access or force users to bypass controls.

Enforced controls and advisory checks play different roles. Policy can block an unapproved destination or enforce a payment threshold; a review model can surface ambiguity and recommend escalation. Advisory checks should never be the only protection against effects that must not occur automatically.

Policy permission is not enough. Runtime controls must contain where data can go and what the agent can affect.

Consequential action boundaries should be explicit

Classify production changes, sensitive-data movement and material external or financial actions as consequential boundaries.

For each boundary, define the enforced limit, required approval and recovery path.

Prefer time-limited grants to standing expansion of authority. Re-evaluate the task after an approved boundary crossing.

Test malicious context, changed capabilities and attempts to bypass the intended boundary.

Control placement and validation

No single control layer can determine that an action is safe. Instructions can improve judgment, runtime isolation can contain execution, resource-side policy can enforce business rules and approval can resolve ambiguity. Test both the intended path and plausible bypasses.

Place controls as close as possible to the effect they limit. Payment actions should enforce thresholds, repository protections should block unsafe changes and egress controls should prevent sensitive data from reaching unapproved destinations.

Validation should show the control working at the boundary. Test normal use and realistic failure, then retain enough evidence to show what was attempted and what occurred.

Model and orchestration	Guide planning and request escalation
Agent and tool boundary	Validate the proposed action against task scope
Resource and destination boundary	Enforce limits close to the affected system or data
Runtime and network boundary	Constrain execution and reachable destinations
Detection and response	Identify unsafe activity and enable intervention

When a task is blocked, operators should know which boundary fired and what evidence would justify an exception. Clear feedback helps prevent brittle systems that push users toward less governed deployments.

Section 03

Securing agent operations

Deployment and evaluation

Deploying an agent is the start of its operational lifecycle, but it's important to continue evaluating its actions and the health of its runtime as it operates. As an agent takes on new capabilities, broader tasks, or more autonomy, it may require additional credentials or authority. Security teams need the same visibility into those changes that they had when the agent was first deployed. As workloads and execution environments evolve, supply-chain trust and posture monitoring also become important, just as they are for other enterprise workloads.

The agent's software and the infrastructure that supports it should often be evaluated as separate components, although some parts of that infrastructure may be opaque to the enterprise. Depending on how the agent is deployed, some controls and the ability to evaluate them will sit with the enterprise, while others may be operated by a model provider platform. Understanding that division can help security teams determine what's directly observable, what the provider is responsible for, and where evaluation should be managed.

Regardless of the deployment model, enterprises should maintain an inventory that identifies the agents operating within their environment and the runtimes used to carry out work. As deployments mature, teams should use workforce identities and available workload identities to inform access decisions. Independent controls should enforce the authority assigned to each task. Workforce identities identify the person or organizational principal behind a task, while workload identities identify the runtime performing it. Those identities may be managed internally or provided through an external identity provider.

Once the agent runtime and its underlying workload can be identified, security teams are better positioned to monitor agent access and behavior, along with the agent's backing infrastructure over the course of its operation. Deployment attributes can inform access decisions alongside signals from individual requests. Meaningful changes to the agent's access or operating environment should trigger a new security review. When a model provider platform operates the relevant controls, teams should understand what evidence it can provide.

Balancing usability with security

As with any new system, deploying agents requires finding the right balance between security friction, user productivity, and organizational trust. Controls that make legitimate work difficult can discourage adoption or push users toward less governed alternatives. When confidence in a deployment is still developing, a clearly defined purpose can help teams limit access to what the agent needs for its work. As teams gain evidence that those limits are effective, they can consider expanding the agent's capabilities.

Before access or autonomy expands, security teams should understand what additional authority is being granted and confirm that it has been approved. Monitoring and recovery paths should still support the change. This allows the organization to give users more flexibility while maintaining visibility into how the agent operates and what it is able to do.

Observability, detection, and response

Once agents have identifiers to which actions and access can be attributed and evaluated, organizations should then be capable of observing those agent runtimes as they perform delegated or autonomous work on behalf of other actors. With this capability, security teams can detect risky behavior from an agent and respond accordingly when work is performed by having clear access authorization boundaries. Agent-native telemetry and access controls in the harness, such as guardian agents that can quickly evaluate and authorize agent access, can also provide meaningful context and a helpful enforcement measure, but traditional security logging and incident review remains essential.

Access gateway telemetry can show what action was attempted with authority, while endpoint telemetry can help determine the state of the agent and workload when an action was performed. Additionally, harness telemetry can often help explain why and to what purpose, what credentials were used and, when used in conjunction with guardian agents, why initial authorization was granted. Using these sources, we can build a clear understanding of intent, action, and posture, with which we can both enforce access and determine response in the case of misaligned or incorrect actions being taken.

At a minimum, collect:

- [ ] Initiating principal, agent identity and task
- [ ] Relevant context and policy, with sensitive content minimized or redacted
- [ ] Tool action and destination
- [ ] Required approvals and enforcement decisions
- [ ] Action result and material effect
- [ ] Correlation with existing security telemetry

Once organizations can gather this telemetry, it can be used to enforce access for agents at time of request, or help provide rich insight during incident response. Response is something that agents are well-equipped to perform, and help facilitate alongside traditional IR tooling. Production platforms should let responding services and security teams stop unsafe work, revoke the authority behind it, contain the affected capability or destination and verify whether data crossed the boundary.

Stopping execution and revoking authority are separate operations. Terminating a task might not invalidate a token already issued to another service, and revoking a credential does not stop work already underway. It's important for response controls to be capable of addressing both. When reviewing an incident, it can be useful for both agents and human responders to present the attempt, policy decision and downstream effect, with the identifiers needed to connect them. This lets responders distinguish an unsafe attempt that was blocked from an unauthorized effect that completed.

Useful evidence does not require indiscriminate collection of content, especially when using agents to evaluate the evidence on a responder's behalf. Preserve the references and decision metadata needed to review actions while minimizing and protecting content that should remain private or out of scope for review. While prompt, context, and additional data can help explain the action, it can create privacy obligations, so it's important to ensure that evaluation doesn't introduce additional risks like data leakage.

In order to act quickly, organizations should enable responders to:

- ✓ Stop running agents
- ✓ Revoke access credentials for those agents
- ✓ Observe actions taken by agents
- ✓ Understand the data sent and received by an agent

Responding to an incident

Treat an agent incident as a failure or misuse that creates, or could plausibly create, an unauthorized effect. To evaluate these incidents, response teams should be able to connect the task and action chain to the resulting impact. This evaluation can help responders stop affected work and separately revoke an agent's access.

Base severity on effective blast radius and completed effect: what changed, what remains active and whether other deployments may be exposed.

A response plan should cover four stages: stop, contain, recover and learn. Stop active work and remove the narrowest risky capability. Repair the affected state, then use the incident to improve policy or testing.

Agents can help responders reconstruct activity and recommend containment, but automated response should be limited to pre-approved, well-bounded containment actions. Human responders should retain judgment over material or ambiguous incidents.

For each production deployment:

Document who responds, how to stop unsafe work and what is required to resume operation.

Exercise response drills before broad autonomy:

including a compromised capability and an attempted sensitive-data transfer.

CISO first-actions checklist

Use this checklist to decide which agents are ready for production or broader deployment. The aim is to establish clear ownership, understand what an agent can affect and ensure security teams can intervene when something goes wrong.

- [ ] **Know your agents.** Maintain an inventory of pilot and production agent deployments that records the accountable owner, business purpose, systems and data each can reach and actions each can take.

- [ ] **Prioritize by potential impact.** Assess each deployment's effective blast radius: what it can affect, which actions can complete without another decision point and how difficult those effects are to reverse.

- [ ] **Set a clear expansion gate.** Before increasing an agent's reach, document credible failure and abuse scenarios, what can influence the agent's decisions and which sensitive actions require approval.

- [ ] **Keep authority scoped to the task.** Give agents only the access and duration needed for the work. Require an independent check before actions such as sending funds, changing records or contacting customers.

- [ ] **Test safeguards under realistic conditions.** Include malicious instructions in retrieved content and inappropriate tool calls. Verify that controls prevent harm and responders can stop the agent or revoke its access.

- [ ] **Reassess meaningful changes.** Review agent deployments when they gain new tools, data access, autonomy or execution environments that could change their impact.

- [ ] **Plan for intervention.** Ensure security teams can see who authorized a consequential action, what the agent did and what changed. They should be able to stop unsafe work and investigate incidents.

Next steps

For CISOs and security teams, secure agent adoption begins with understanding what each agent is authorized to do, which systems and data it can access, and what could happen if its behavior is mistaken or manipulated. A credential or tool connection is not permission to take every available action. Agents should operate within boundaries defined by their assigned tasks, with independent controls that limit consequential effects and preserve clear accountability.

Begin by identifying existing agent deployments, assigning accountable owners, and assessing their potential impact. Establish which controls are managed by the enterprise or its providers, where additional approval is required, and how security teams will detect and respond to unsafe activity. Make sure that agents have access to revocable credentials that are short-lived when possible, and that longer-lived credentials are securely managed and used outside of the agent's context window.

Delegate the work that agents are performing in a way that gives security teams the tools they need to observe and enforce access depending on the task, the agent and its role.

Define the limits of their authority through enforceable controls around sensitive data, critical systems, and consequential actions.

Remain accountable for the outcome by maintaining clear ownership, operational visibility, and the ability to intervene when necessary.

With these foundations in place, security teams can help enable users to build trust in their agents and take on increasingly ambitious and productive goals.