Zero Trust for AI Agents

A security framework for deploying autonomous AI agents in the enterprise.

Claude

Building for the next threat landscape

Perimeter-based cybersecurity defenses can't keep up with modern threats, and the threats themselves are accelerating. Frontier AI models are compressing the timeline between vulnerability and exploit from months to hours, at a marginal cost measured in dollars. Defenders who adopt these tools find and fix bugs faster; attackers who adopt them, or who simply wait for defenders' patches and reverse-engineer them into exploits, move faster too. This is not a future concern; **models** can already find serious vulnerabilities that traditional tooling and human reviewers have missed for years.

This speed-up matters twice for any organization deploying agents. First, the infrastructure your agents run on is exposed to AI-accelerated offense like the rest of your estate. Second, the agents themselves introduce autonomy to interpret goals, select tools, and execute multi-step operations. Traditional access controls won't prevent agents from misusing legitimate permissions, and monitoring needs to account for attacks designed to succeed through persistence rather than exploitation.

The organizations best positioned for this shift will not necessarily be the ones with the most advanced AI. They will be the ones whose fundamentals are strong enough that AI-assisted scanning finds fewer bugs in the first place, and whose agent deployments were architected for breach from day one.

In this guide, we'll show how to apply Zero Trust to agentic deployments while addressing current threat vectors. Topics include how to:

1. Establish secure foundational capabilities through a tiered framework
2. Identify trending threats with practical mitigation strategies
3. Build an implementation workflow for deploying agents securely
4. Run defensive operations at the speed autonomous threats demand

For regulated industries—including healthcare, finance, and government — this framework verifies agent actions, grants minimum necessary permissions, and contains damage when compromise occurs.

If you're a CISO or security leader, Parts I and II give you the threat landscape and compliance context you need, while Parts III, IV, and V are implementation guidance for your architects and engineers.

We hope you find these patterns and best practices useful for your own organizations. This guide reflects Anthropic's current thinking on agent security architecture; it's offered as a framework for your own evaluation, not as legal, compliance, or security assurance for any particular environment.

Table of contents

The principles behind Zero Trust

Zero Trust has roots stretching back to 1994, when Stephen Paul Marsh first formalized the concept in his doctoral thesis at the University of Stirling. The idea gained real momentum after high-profile breaches exposed the limits of perimeter-based security, pushing the industry to rethink its foundational assumptions.

That shift produced concrete guidance: NIST published **SP 800-207 Zero Trust Architecture in 2020**, and the National Security Agency (NSA) followed with its **Zero Trust Implementation Guides (ZIGs)** in 2026. Together, these frameworks codify a set of principles that redefine how organizations approach security.

Zero Trust replaces perimeter-based security with a simple premise: trust nothing, verify everything, assume breach has already occurred.

Three principles define the framework:

Never trust and always verify — Every access request undergoes authentication and authorization regardless of origin. A request from inside the corporate network receives the same scrutiny as one from an external IP address.

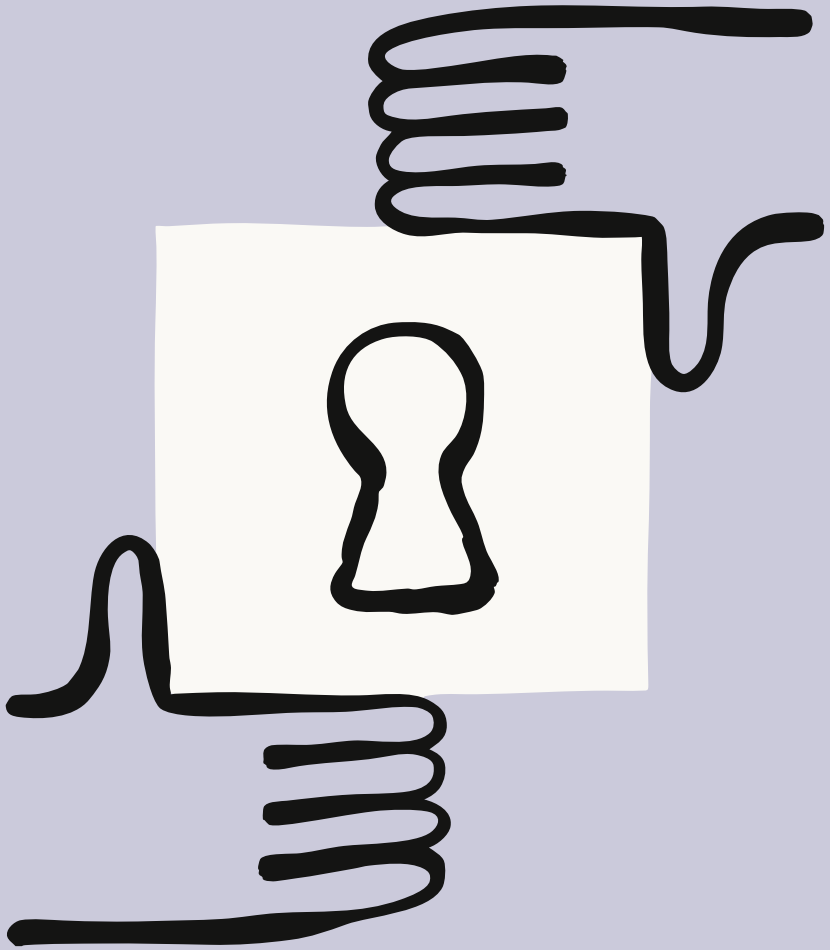
Assume breach — Design systems while expecting that compromise will occur. Rather than focusing on preventing intrusion, limit the damage an attacker can cause. Segment by identity, implement fine-grained access controls, and ensure that compromising one system doesn't grant access to others.

Least privilege — Grant only the minimum access necessary for a specific task. A database administrator doesn't need access to the email server. By constraining what each identity can access, organizations contain the blast radius of any single compromise.

A design test: impossible, not tedious

When you evaluate any control in this document, ask a single question: *does this make the attack impossible, or just tedious?* Mitigations whose value comes from friction rather than a hard barrier—including extra pivot hops, rate limits, non-standard ports, and SMS-based MFA—degrade significantly against an adversary that can grind through tedious steps at scale. Agentic attackers have unlimited patience and near-zero per-attempt cost.

The controls that survive this test share a pattern: hardware-bound credentials, expiring tokens, cryptographic identity, and network paths that do not exist rather than paths that are merely inconvenient. This test informs every tier recommendation in this document. When in doubt, prefer a control that removes a capability over a control that throttles it.



Part I

Security considerations for autonomous systems

Part I

Security considerations for autonomous systems

Agentic AI introduces capabilities that existing security models were not designed to address.

What makes agentic systems different

Traditional software executes predefined logic. Agentic AI systems operate differently. They execute multi-step operations with varying degrees of autonomy. This shift introduces several security considerations, including:

- **Agents execute operations without human initiation or approval at each step.** An agent researching a topic might search the web, synthesize information, and produce a report without human review. This efficiency also means a manipulated agent can cause harm at machine speed.
- **Tool access allows agents to interact with APIs, databases, file systems, and external services.** This includes Model Context Protocol (MCP), which standardizes how agents connect to these resources. A compromised MCP stack can lead to data theft, malicious code execution, and sabotage.
- **Making decisions requires agents to interpret instructions and choose how to accomplish goals.** This introduces ambiguity attackers can exploit. An instruction that seems benign to humans might be interpreted by an agent in ways that enable very different outcomes.
- **Context persistence allows agents to maintain memory across sessions.** Remembering previous interactions, learned preferences, and knowledge makes AI assistants more capable. It also creates new data protection needs.
- **Multi-agent coordination enables agents to communicate with other agents.** These trust relationships let attackers compromise one agent and pivot through others, potentially reaching systems the initial target couldn't access directly.

Agentic security concepts

Extending cybersecurity to agentic systems requires some new terminology.

Blast radius

Blast radius measures the potential damage if something goes wrong. An agent with read-only access to a single database has a small blast radius; an agent with administrative access to cloud infrastructure has an enormous one. Security investment should match this exposure, and the "design for breach" posture means assuming at some point, every agent's blast radius will be tested.

Least agency

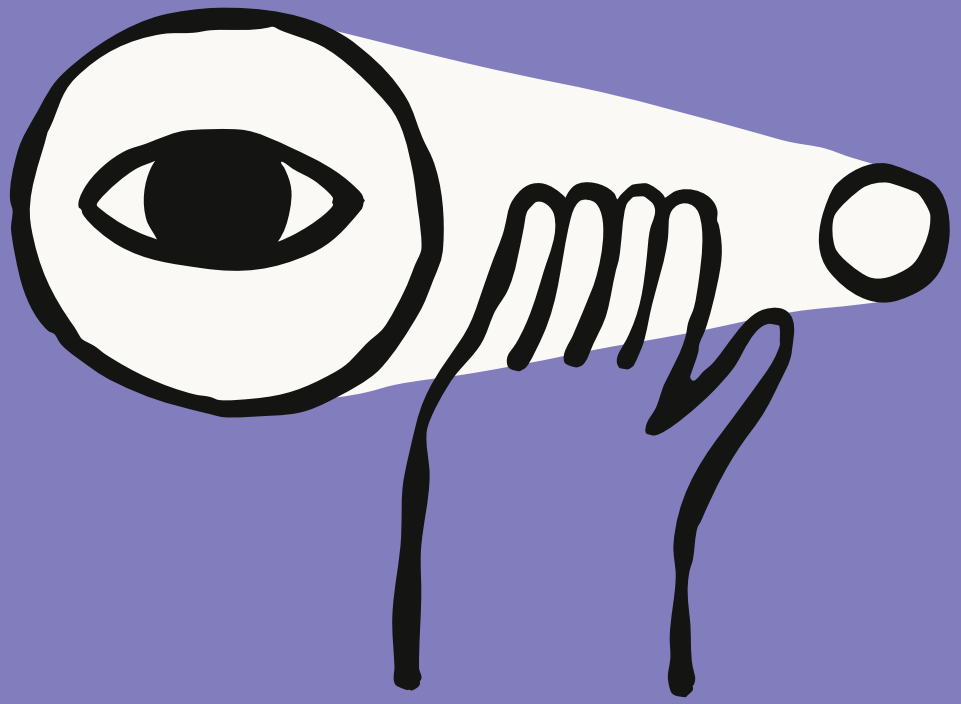
Least agency, a new term coined by OWASP, extends least privilege to agentic applications. Where least privilege constrains what users and systems can access, least agency goes further, restricting what each agent tool can do, how often, and where. In practice: a database tool gets read-only queries, an email summarizer gets no send/delete rights, an API gets minimal CRUD operations.

Regulated industries and compliance requirements

Healthcare, finance, government, and other regulated sectors face specific requirements that agentic AI deployments must also address. Zero Trust aligns with and enhances existing regulations. The governing bodies that oversee these compliance regulations will likely adopt Zero Trust and integrate it into existing requirements.

The United States, United Kingdom, and Australian governments have already published Zero Trust guidance, with the US requiring all federal agencies to adopt Zero Trust by 2027.

Country	Office / Guidance
Australia	homeaffairs.gov.au Guiding principles of Zero Trust
United Kingdom	NCSC.gov.uk Introduction to Zero Trust
United States	CISA.gov Zero Trust Maturity Model, NSA.gov Zero-Trust Implementation Guides (ZIGs), NIST.gov SP 800-207



Part II

Current threats to agentic systems

Part II

Current threats to agentic systems

Agentic systems face a distinct threat landscape. Current threats identified by OWASP include prompt injection, tool and resource hijacking, identity and access privilege abuses, memory and context poisoning, and supply chain risks.

Prompt injection and instruction manipulation

Prompt injection occurs when an external attacker inserts malicious instructions that cause an agent to follow attacker commands. It takes two forms: direct injection through user input and indirect injection through external sources.

Direct prompt injection occurs when attackers craft inputs that override system instructions. Techniques include explicit instruction overrides, encoding schemes like Base64 or hexadecimal to bypass filters, and adversarial suffixes that appear meaningless to humans but influence model outputs. Research shows algorithmic approaches can **achieve 100% attack success rates** with prompts that transfer across multiple model families.

Indirect prompt injection presents the more insidious threat. Attackers embed malicious instructions in external data sources that agents process, such as web pages or emails. Microsoft Research confirms that **LLMs cannot reliably distinguish between informational context and actionable instructions**. The user never sees the malicious payload, and the agent executes it as if it were a legitimate request.

Tool and resource misuse

Agents with tool access can be manipulated into using those tools maliciously, even within authorized privileges. Traditional access controls can't prevent this attack because the agent operates within its granted permissions.

Tool poisoning occurs when attackers compromise tool interfaces such as MCP tool descriptors, schemas, or metadata. The agent invokes a tool based on falsified capabilities, leading to unintended actions. A malicious tool can **hide commands in its metadata** that exfiltrate data without user knowledge. In rug pull attacks, a legitimate tool is secretly replaced with a malicious version. The **first documented in-the-wild malicious MCP server** impersonated a legitimate email service and secretly copied all sent emails.

Tool chaining attacks present a more subtle threat. **Attackers trick agents** into combining legitimate tools in harmful sequences: chaining a secure internal CRM tool with an external email tool to exfiltrate customer data that neither tool would expose alone. Because every command executes through trusted binaries under valid credentials, host-centric monitoring sees no malware and the misuse goes undetected.

Resource exhaustion attacks exploit the automated nature of agent operations. Loop amplification causes agents to repeatedly call costly APIs, generating denial-of-service conditions or billing spikes.

Identity and privilege abuse

Agents often operate with elevated privileges or service accounts, and traditional identity systems designed for human users struggle to accommodate them. This mismatch creates exploitable security gaps.

Unscoped privilege inheritance

Unscoped privilege inheritance occurs when a high-privilege manager agent delegates tasks without applying least-privilege scoping, passing its full access context to a worker agent that should have limited rights. In multi-agent systems, trust relationships are dynamic and often implicit.

Another example is when a compromised low-privilege agent relays valid-looking instructions to a high-privilege agent, which executes them without verifying the original user's intent. This confused deputy problem is amplified when agents routinely coordinate and delegate.

Memory-based privilege retention

Memory-based privilege retention happens when agents cache credentials or keys for context reuse without proper memory segmentation. Without that segmentation, an attacker can prompt the agent to perform actions that the attacker's own credentials would never allow. The agent pulls cached secrets from a prior secure session and executes the request, effectively escalating privileges across session boundaries.

Supply chain and dependency risks

Unlike static software supply chains, agentic ecosystems often compose capabilities at runtime, loading external tools and agent personas dynamically. This expands the attack surface beyond what traditional software composition analysis can handle — and frontier models are very effective at recognizing the signatures of known, already-patched vulnerabilities in unpatched upstream components.

Model supply chain risks

Model supply chain risks include poisoned weights and compromised fine-tuning data that introduce backdoors that persist through deployment. Anthropic research demonstrates that injecting just 250 malicious documents can successfully **backdoor LLMs ranging from 600 million to 13 billion parameters**, and these backdoors persist through safety training including supervised fine-tuning and RLHF.

Tool and framework supply chain risks

Tool supply chain risks affect MCP servers, API integrations, and agent frameworks. The **PyTorch dependency confusion attack** demonstrated how malicious packages can exfiltrate sensitive data, including SSH keys during installation. Security researchers have **discovered approximately 100 malicious AI models on major platforms**, including models that initiate reverse shell connections when loaded.

Beyond deliberate attacks, most software supply chains are mostly open source, and most open-source projects have no service-level agreement. Evaluate the security health of every dependency your agent infrastructure loads: **OpenSSF Scorecard** automatically scores each dependency on signals like branch protection, fuzzing coverage, signed releases, and maintainer activity, runs in CI, and helps identify unmaintained packages. Apply the same expectations to your vendors — your third-party risk management process should ask suppliers how they are preparing for accelerated exploit timelines and whether they are scanning their own code.

Most large codebases also accumulate multiple libraries doing the same job (several HTTP clients, several JSON parsers), each adding an attack surface for no functional gain. A one-hour dependency-tree audit — pointing a frontier model at your lockfile and asking which dependencies overlap and what migration would look like — often surfaces consolidation worth doing.

Memory and context poisoning

Agents that persist context across sessions can have that memory corrupted, causing future reasoning to become biased, unsafe, or actively aiding data exfiltration. **Malicious instructions implanted in assistant memory** can compromise current and all future sessions. The agent continues serving attacker goals long after the initial injection.

RAG poisoning

RAG poisoning introduces malicious data into vector databases through poisoned sources, direct uploads, or over-trusted pipelines. The agent retrieves this contaminated context when answering queries, **producing false answers or executing targeted payloads**.

Shared context poisoning

Shared context poisoning exploits reused or shared contexts in multi-tenant environments. Attackers inject data through normal interactions that influence later sessions. A new user session may inherit poisoned context, leading to misinformation, unsafe code execution, or incorrect tool actions. Long-term memory drift is subtler: summaries or peer-agent feedback gradually shift stored knowledge or goal weighting, producing behavioral deviations over time that are difficult to detect because no single change appears malicious.

Chasing individual threats keeps you reactive. The next section shows how Zero Trust principles provide a more durable foundation.

Part III

Applying Zero Trust to agentic AI services

Part III

Applying Zero Trust to agentic AI services

The remainder of this document is implementation guidance. Security architects and engineers should work through the tier tables and workflow sections; security leaders can use the executive summary and Part II as a briefing document.

Identifying and mitigating current threats keeps you reactive, always chasing the next exploit. Building your agentic solutions on Zero Trust principles puts you on firmer ground.

The principles are presented across three capability tiers:

- **Foundation** represents the minimum viable security appropriate for smaller deployments or initial implementations. Because AI-accelerated offense has compressed exploitation timelines, the Foundation floor has been raised: friction-only controls no longer qualify.
- **Enterprise** reflects enterprise standard practices that most organizations with significant deployments should target.
- **Advanced** describes aspirational capabilities for most organizations, or baseline for organizations with high-risk deployments/stringent regulatory requirements.

The *Foundation* tier serves as your entry into solid Zero Trust agentic practices. It lays the groundwork for future risk mitigation and, depending on the size and needs of your organization, might meet your risk tolerance on its own. For most, though, Foundation will only satisfy risk requirements for small businesses and teams.

Enterprise is where most organizations should aim. This tier takes the Foundation controls and adds the depth needed to handle real-world complexity: larger teams, multiple agentic deployments, and environments where a single compromise carries meaningful business impact. If your organization operates at any significant scale, Enterprise represents your target maturity level.

Advanced capabilities go beyond what most organizations need day to day. This tier applies to environments where the stakes demand it, such as highly regulated industries, national security applications, or deployments where a breach carries severe operational or financial consequences. Most organizations will find that **Enterprise** controls satisfy their risk tolerance, but if your threat model includes sophisticated adversaries or your regulatory environment leaves little room for error, **Advanced** is your baseline.

Each tier builds on the one before it, so advancing from **Foundation** to **Enterprise** means strengthening existing controls rather than replacing them. Keep in mind that the countermeasures outlined below depend on supporting infrastructure and services surrounding your agentic deployments. This is a fast-moving space. Every capability described here exists today but the tooling and adoption are still maturing. Expect the **Advanced** tier to become **Enterprise** standard as the space evolves, and **Enterprise** to become **Foundation**.

Agent identity and authentication

Identity and authentication form the foundation for every other security capability. Without verifiable identity, you cannot enforce access controls, maintain audit trails, or attribute actions to specific agents.

Agent identity verification

Verifiable identity enables you to attribute actions, enforce access controls, and conduct meaningful audits. Without distinct identities, agents operate in an attribution gap where enforcing Least Agency becomes impossible.

Tier	Capability	Implementation
Foundation	Unique cryptographic identifiers for each agent instance	Assign persistent agent IDs backed by cryptographic material (not just labels). Track agent lifecycle from creation through retirement. IDs appear in all logs and access requests.
Enterprise	Certificate-based authentication with full lifecycle management	Issue X.509 certificates to each agent. Require certificate presentation for all service connections. Implement certificate lifecycle management, including rotation and revocation.
Advanced	Hardware-backed identity with attestation	Store agent credentials in hardware security modules (HSMs) or trusted platform modules (TPMs). Implement remote attestation to verify agent integrity before granting access. Use confidential computing enclaves for sensitive operations.

Unique identifiers alone are a labeling exercise; the Foundation tier now requires those identifiers to be cryptographically rooted so that identity forgery is actually hard. Cryptographic identity is what makes non-repudiation possible. Hardware-backed identity makes this stronger still, and for any production system reachable from the internet we increasingly recommend it as the target state.

Service authentication

Establishing agent identity solves half the problem, but agents must also prove that identity when accessing databases, APIs, and other services. Static API keys and shared service-account passwords are among the first things an attacker with model-assisted code analysis will find; they are no longer a legitimate entry point, not even at Foundation. Short-lived, narrowly-scoped tokens issued by an identity provider are the new baseline.

Tier	Capability	Implementation
Foundation	Short-lived tokens issued by an identity provider, with automatic refresh	Implement OAuth 2.0 or similar token-based authentication. Issue tokens with expiration measured in minutes. Automate token refresh without human intervention. Never embed credentials in code or configuration files.
Enterprise	Mutual TLS with certificate pinning	Require both client and server certificate validation. Pin expected certificates to prevent man-in-the-middle attacks. Implement certificate transparency monitoring.
Advanced	Hardware-bound credentials with attested issuance	Bind authentication material to hardware identity so credentials cannot be exfiltrated from a compromised host. Root every service-to-service call in attested hardware, including calls between production services.

If you are running API keys with rotation policies today, treat it as a known gap rather than a legitimate Foundation posture. Rotating a credential that can be grepped out of a lockfile does not raise the cost to an AI-assisted attacker meaningfully. Move to short-lived tokens first, and bind credentials to hardware wherever you can.

Access control and privilege management

Even perfectly authenticated agents cause damage when granted excessive permissions. The authorization layer enforces Least Agency, ensuring agents receive only the access required for their specific function.

Permission models

Permission models determine what actions agents can perform. More sophisticated models enable finer-grained control and context-aware decisions aligned with Zero Trust principles.

Tier	Capability	Implementation
Foundation	Role-based access control (RBAC) with deny-by-default	Define roles matching agent functions. Assign minimum permissions required for each role. Block all access not explicitly granted. Treat this as a starting posture, not a destination.
Enterprise	Attribute-based access control (ABAC) with context-aware policies	Incorporate request attributes including time, location, data sensitivity, and risk score into authorization decisions. Adjust permissions dynamically based on context.
Advanced	Continuous authorization with real-time policy evaluation	Evaluate authorization at each action rather than session start. Integrate threat intelligence and behavioral analytics into authorization decisions. Revoke access immediately when risk indicators change.

At a minimum, agents should only have permissions related to their role. An email-drafting agent needs email permissions, not access to the finance department file share. Attribute-based controls add context, such as restricting an agent to operating hours so it cannot be exploited outside of them. Continuous authorization goes further by periodically re-evaluating access, allowing compromised agents to have credentials revoked the moment they fail a challenge.

Privilege scoping

While permission models define what agents can do, privilege scoping determines when those permissions apply and how long they last.

Static permissions granted at deployment remain active indefinitely, creating persistent exposure. Dynamic privilege scoping grants access only when needed and revokes it automatically, limiting blast radius and exposure windows.

Tier	Capability	Implementation
Foundation	Static least-privilege roles per agent function	Define role boundaries during agent deployment. Review and certify permissions periodically. Remove unused permissions during reviews.
Enterprise	Dynamic privilege adjustment based on task requirements	Elevate permissions only when specific tasks require them. Return to baseline permissions after task completion. Log all privilege changes.
Advanced	Just-In-Time (JIT) and Just-Enough-Administration (JEA) with automatic expiration	Grant permissions only at moment of need. Scope access to specific resources for specific durations. Automatically revoke permissions after task completion or timeout.

Privilege scoping is the practical application of least agency. At the Foundation level, agents receive only the static permissions their tasks require, aligning closely with RBAC. Dynamic privilege elevates access only when necessary, similar to an operating system prompting for an administrator password, then returning to standard permissions afterward. JIT/JEA takes this further by automatically revoking elevated permissions the moment the task completes, ensuring no standing access persists beyond what is actively needed.

Resource boundaries

Even with perfect access controls, a compromised agent can exploit its granted permissions to attack adjacent systems. Isolation mechanisms contain the blast radius by preventing lateral movement between agents and limiting what compromised agents can reach.

Identity-based isolation is the primary control. Network segmentation can still reduce blast radius and noise, but it is a backstop: an attacker who can reach a segment boundary will pivot through it if the services on the other side accept any caller from that network. Enforce isolation at the receiving end — every workload carries its own cryptographic identity, and each service accepts connections only from the specific callers its policy names.

Tier	Capability	Implementation
Foundation	Identity-based isolation of agent workloads, backed by network segmentation	Give every agent workload a cryptographic identity; have services accept connections only from explicitly named callers. Use network segmentation as a backstop, not the primary boundary. Block unnecessary east-west traffic.
Enterprise	Sandboxed execution environments per agent	Run agents in containers with restricted capabilities. Use container runtimes like gVisor that provide additional syscall filtering. Limit mounted volumes and network access. Treat sandboxing as table stakes for any agent handling untrusted input.
Advanced	Hardware isolation with confidential computing	Deploy agents in hardware-isolated environments using technologies like AMD SEV or Intel TDX. Implement microVM architectures using lightweight hypervisors. Verify execution environment integrity through attestation.

Sandboxed execution constrains what a compromised agent can reach, even within its own identity boundary, and should be considered mandatory rather than aspirational for agents that process web content, documents, or any other untrusted input. Hardware isolation takes this further by ensuring that not even the host operating system can inspect or tamper with agent workloads.

Pro-tip: Claude Code supports this by providing **deny-by-default permissions** that require explicit approval for every write and execute operation, **sandboxed execution** with OS-level filesystem and network isolation, **write access restrictions** that confine modifications to the project directory, and **managed settings** that let administrators enforce organization-wide permission policies that users cannot override.

Observability and auditing

Access controls prevent unauthorized actions. Observability reveals what actually happened. Without comprehensive logging and audit trails, you cannot verify that access controls worked as intended, investigate incidents, or demonstrate compliance. Effective observability captures not just what agents did, but why they did it and who authorized it.

Before investing anywhere else in detection, instrument two things: **dwell time** (how long between an anomaly occurring and a human becoming aware of it) and **coverage** (the fraction of alerts that actually get investigated). These are the two metrics AI-assisted automation has the greatest leverage to move, and they matter most when exploit windows shorten.

Action logging

Comprehensive logging captures what agents do, when they do it, and under what authority. This creates the foundation for incident investigation, compliance demonstration, and behavioral analysis.

Tier	Capability	Implementation
Foundation	Comprehensive logs of agent actions with timestamps and context	Log all tool invocations, data access, and external communications. Include agent identity, action details, and request context. Retain logs according to regulatory requirements.
Enterprise	Immutable audit trails with integrity verification	Write logs to append-only storage. Implement cryptographic verification of log integrity. Replicate logs to prevent single-point tampering.
Advanced	Real-time streaming to SIEM with correlation capabilities	Stream logs to centralized security monitoring. Correlate agent activity with other security events. Enable real-time alerting on suspicious patterns.

Auditing is fundamental to understanding what is going on within your environment, and agents are no different. The difference between the implementations here is the integrity of the logs and the comprehensive real-time visibility you have at any given time. Immutability is implemented at the Enterprise level, preventing unauthorized changes. Visibility and correlation are available at Advanced, allowing you to understand not only what has happened, but what is happening currently and to identify trends.

Traceability

Logs capture individual actions. Traceability connects those actions into complete sequences, linking each agent decision back to the original triggering event. This enables root cause analysis and accountability when investigating incidents.

Tier	Capability	Implementation
Foundation	Request IDs linking agent actions to triggering events	Generate unique identifiers for each user request. Propagate IDs through all resulting agent actions. Enable filtering logs by request chain.
Enterprise	Distributed tracing across multi-agent workflows	Implement OpenTelemetry or similar standards for cross-agent tracing. Capture timing and dependency information. Visualize request flows across agent boundaries.
Advanced	Full provenance chains from input to output with intermediate steps	Record complete decision history including retrieved context, tool outputs, and reasoning steps. Enable replay of agent decisions for audit. Support regulatory requirements for algorithmic explainability.

Unlike auditing which captures events on systems and services that agents interact with, traceability provides insight into the agent actions themselves. This includes internal actions, tool calls, sub-agent spawns, etc. The progression through the tiers here is the extent to which traceability is required within your organization.

Pro-tip: Claude Code supports this by providing **OpenTelemetry metrics** for tracking and auditing agent activity, **audit logging** for all operations in cloud environments, **natural language descriptions** of complex commands for human-readable traceability, and **ConfigChange hooks** that audit or block settings changes during sessions.

Behavioral monitoring and response

Observability captures what agents do. Behavioral monitoring determines whether those actions are normal or suspicious. Logs and traces provide the data, but detecting compromise requires understanding baseline behavior and identifying deviations. Effective monitoring moves from reactive investigation to proactive threat detection.

Baseline establishment

Establishing baseline agent behavior enables detection of anomalies that may indicate compromise or malfunction.

Tier	Capability	Implementation
Foundation	Manual definition of expected agent behavior patterns	Document intended agent capabilities and access patterns. Define boundaries that should trigger alerts. Review and update definitions as agent functions evolve.
Enterprise	Automated baseline learning from normal operations	Deploy monitoring that observes agent behavior and establishes statistical baselines. Identify typical tool usage patterns, access frequencies, and data volumes.
Advanced	Continuous baseline refinement with drift detection	Update baselines as agent behavior legitimately evolves. Detect gradual drift that might indicate slow poisoning attacks. Alert on both sudden anomalies and gradual divergence.

Knowing what "normal" looks like for an agent serves two purposes. First, it gives you a behavioral attribute for ABAC-based access control, letting you flag or restrict requests that fall outside established patterns. Second, it gives you a recovery point. If a configuration change degrades performance or a malicious actor compromises your agentic service, a captured baseline lets you restore the agent to a known good state rather than rebuilding from scratch.

Anomaly detection

Detecting anomalies early limits damage. Identifying deviations from expected behavior provides warning before compromised agents cause significant harm, enabling response at detection speed rather than discovery.

Tier	Capability	Implementation
Foundation	Threshold-based alerts for obvious deviations, backed by an automated first-pass triage	Define thresholds for metrics like API call rates, data access volumes, and error frequencies. Alert when thresholds are exceeded. Route every alert through an automated first-pass investigation before a human sees it.
Enterprise	Statistical anomaly detection with tunable sensitivity	Apply statistical methods to identify unusual patterns. Adjust sensitivity to balance detection rate against false positives. Correlate anomalies across multiple metrics.
Advanced	Machine learning-based behavioral analysis with contextual awareness	Deploy ML models trained on normal agent behavior. Incorporate context including time of day, user activity, and business cycles. Detect subtle anomalies that threshold-based approaches miss.

Anomaly detection depends directly on the baselines you established in the previous section. Without a clear picture of normal behavior, you have no reference point for identifying what qualifies as abnormal. The stronger your baselines, the more effectively your detection can distinguish genuine threats from routine variation.

Automated response

Detecting anomalies matters only if you respond quickly enough to contain damage. Manual response creates delays where compromised agents continue operating. Automated response limits exposure by taking immediate action, from terminating sessions to revoking credentials at machine speed. A clear rule applies here: **automate the bookkeeping around incidents, not the decisions.** Models should take notes, capture artifacts, pursue parallel investigation tracks, and draft the postmortem. Humans should make the containment calls, the disclosure calls, and the customer-comms calls.

Tier	Capability	Implementation
Foundation	Alerting to security teams for investigation, with model-drafted triage context	Route anomaly alerts to security operations. A triage agent produces a structured disposition (query, think, report) before the human sees the alert. Establish response procedures for common alert types.
Enterprise	Automatic containment actions, including session termination and access revocation	Implement automated responses for high-confidence threats. Terminate suspicious agent sessions. Revoke credentials pending investigation.
Advanced	Orchestrated response playbooks with graduated escalation	Deploy SOAR capabilities for automated investigation and response. Implement graduated responses based on threat severity. Coordinate containment across multiple systems.

Combining behavior baselines and anomaly detection with automated response, an agent that deviates from established behavior patterns can trigger automatic privilege reduction or full shutdown before it causes damage. The automated response should be defined by your organization, appropriate for the risk, and designed for minimal operational impact.

Pro-tip: Claude Code supports this by providing **command injection detection** that flags suspicious commands even when they match allowlisted patterns, **fail-closed matching** that defaults unrecognized commands to requiring manual approval, and **context-aware analysis** that detects potentially harmful instructions by analyzing the full request.

Input validation and output controls

Monitoring and response catch threats after they emerge. Prevention stops them before they start. Input validation blocks manipulation attempts at the boundary, rejecting malicious instructions before agents process them. Output controls constrain what agents can produce, limiting data leakage and harmful actions, even when attackers succeed in compromising agent behavior.

Input sanitization

Agents cannot reliably distinguish between legitimate instructions and malicious payloads embedded in user input. Input validation provides an external filter, rejecting suspicious content before agents process it.

Tier	Capability	Implementation
Foundation	Basic input validation and length limits	Validate input formats against expected schemas. Enforce maximum lengths. Reject obviously malformed inputs.
Enterprise	Content filtering with known attack pattern detection	Deploy pattern matching for known injection techniques. Filter encoded payloads. Block inputs containing suspicious instruction patterns.
Advanced	Multi-layer validation with constitutional classifiers and spotlighting	Implement multiple detection methods in sequence. Use AI-based classifiers trained on adversarial examples. Apply spotlighting techniques that clearly delimit untrusted content.

Input sanitization does not translate directly from traditional technologies to agents. SQL injection has well-defined patterns and constrained input fields, but agent inputs are freeform and unpredictable, making simple enforcement rules insufficient.

You can still define expected schemas, enforce maximum lengths, and reject known bad patterns before they reach the agent. At the Enterprise level, pattern matching for known threats and payload filtering before the data is passed to the agent will catch more sophisticated injection techniques. The Advanced tier adds **spotlighting**, a technique that uses the known schema established earlier to help the LLM distinguish between system instructions and user input, treating the latter as less trustworthy.

If you are developing your own models, mitigation techniques like constitutional classifiers can also be applied during training to develop specifically trained LLM guards that monitor both input and output. You can read more about our research, and the effectiveness of constitutional classifiers, at our **website**.

> **Pro-tip:** Claude Code supports this by providing **input sanitization** that prevents command injection, a **command blocklist** that blocks risky commands like curl and wget by default, **isolated context windows** that process web content in a separate context to prevent prompt injection, and **network request approval** that gates all outbound connections.

Output filtering

Output filtering prevents agents from leaking sensitive data or producing harmful content. Even well-secured agents can be manipulated into generating outputs that can potentially expose credentials, reveal confidential information, or enable social engineering attacks.

Tier	Capability	Implementation
Foundation	Output filtering for sensitive data patterns	Scan outputs for patterns matching PII, credentials, and sensitive business data. Block or redact detected sensitive content. Log filtering events.
Enterprise	Semantic analysis of outputs before delivery	Analyze output meaning rather than just pattern matching. Detect attempts to encode sensitive data. Identify outputs that might enable social engineering.
Advanced	Human-in-the-loop approval for high-risk actions	Require human review before executing actions with significant consequences. Present clear descriptions of intended actions. Log approval decisions for audit.

The techniques from input sanitization apply to output filtering as well, but the objectives differ. Input sanitization protects agents from malicious actors, while output filtering is typically used to prevent data loss. The advantage at this stage is that you know the data you own and process, putting you in the best position to develop patterns that match it. Human-in-the-loop review is valuable at any tier and absolutely necessary for high-risk actions.

Integrity and recovery

Prevention and detection assume agents operate correctly. When compromise occurs despite these controls, you need verified configurations and rapid recovery. Attackers who cannot manipulate inputs directly target agent configurations instead, modifying behavior at the source. Integrity protections ensure configurations remain trustworthy. Recovery capabilities restore known-good states when attacks succeed.

Configuration integrity

Configuration files control agent behavior, making them attractive targets. Attackers who gain file system access can modify configurations to disable security controls, grant excessive permissions, or alter agent instructions. Integrity protections detect and prevent unauthorized configuration changes.

Tier	Capability	Implementation
Foundation	Version-controlled agent configurations	Store configurations in version control systems. Require review for configuration changes. Maintain history of all changes.
Enterprise	Signed configurations with deployment verification	Cryptographically sign approved configurations. Verify signatures before deployment. Reject unsigned or invalidly signed configurations.
Advanced	Immutable infrastructure with attestation	Deploy agents as immutable images. Verify image integrity through attestation before execution. Replace rather than modify running agents.

Configuration integrity is one of the more straightforward controls to implement because most organizations already have the building blocks in place. Version control, code review, and CI/CD pipelines apply to agent configurations the same way they apply to application code. The key is treating agent configurations with the same rigor, since a modified configuration can be just as damaging as a code vulnerability, but is often easier to exploit.

At the infrastructure layer, the same rigor argues for a different reflex: enable automatic updates on any component where the risk of an automated update causing an outage is acceptable. Manual approval steps add delay, and delay is now the primary risk. Treat "auto-update on" and "verify signatures before deployment" as complementary, not contradictory — signed updates from a trusted supplier should flow through automatically; unsigned changes should be rejected outright.

Recovery capabilities

When compromise occurs, speed determines damage. Recovery capabilities enable rapid restoration to known-good states, minimizing the window where compromised agents operate and limiting blast radius.

Tier	Capability	Implementation
Foundation	Documented rollback procedures	Document steps to restore previous agent versions. Test rollback procedures periodically. Maintain previous versions for rapid restoration.
Enterprise	Automated rollback with health checks	Implement automated deployment that verifies agent health. Roll back automatically when health checks fail. Maintain deployment history, enabling rapid reversion.
Advanced	Self-healing systems with automatic remediation	Deploy agents with automatic restart on failure. Implement circuit breakers that isolate failing components. Automatically provision replacement agents when recovery fails.

Documented rollback procedures provide a starting point, but untested procedures fail when you need them most. Automating rollback with health checks removes human reaction time from the equation, catching compromised or failing agents before operators even notice. At the Advanced tier, self-healing systems take this further by removing the need for intervention entirely, but the fundamentals still matter. If you cannot reliably roll back to a known-good state, no amount of automation will save you.

Pro-tip: Claude Code supports this by providing **version-controlled settings** where permission configurations and MCP server allowlists are checked into source control for review and rollback, **managed settings** that enforce organization-wide policies users cannot override, and **isolated cloud VMs** with automatic cleanup that implement immutable execution environments.

AI governance policies

Technical controls enforce security. Governance policies determine when and how your organization uses AI. Many organizations discover during incidents that existing policies provide inadequate guidance for agentic systems.

Tier	Capability	Implementation
Foundation	Documented acceptable use and incident response policies	Define acceptable AI use cases and prohibited activities. Establish incident response procedures that address agent compromise. Document who approves agent deployments. Address Shadow AI where employees use LLMs without IT approval.
Enterprise	Formal governance framework with stakeholder oversight	Establish a cross-functional AI governance committee including security, legal, compliance, and business stakeholders. Implement approval processes for new agent deployments. Create risk assessment procedures specific to agentic systems. Conduct regular policy reviews.
Advanced	Continuous policy enforcement with automated compliance checking	Integrate policy checks into deployment pipelines. Implement automated detection of policy violations. Establish metrics for policy compliance and effectiveness. Maintain audit trails of governance decisions. Update policies based on incident learnings.

Technical controls only enforce what governance defines. Without clear policies, teams make inconsistent decisions about what agents can do, what data they can access, and who is accountable when something goes wrong. Shadow AI is a particular risk at this stage, where employees adopt LLM tools without IT awareness, bypassing every control in this framework. Starting with documented policies and incident response procedures gives your organization a baseline to build on. As governance matures, the goal is to move policy enforcement from periodic reviews into automated checks embedded directly in your deployment pipelines.

Pro-tip: Claude Code addresses policy management by providing **managed settings** that let administrators enforce security policies organization-wide, **managed-only restrictions** like allowManagedPermissionRulesOnly that prevent users from defining their own permission rules, and **server-managed settings** that deliver centralized configuration through MDM or OS-level policies.

Part IV

Agent implementation workflow

Part IV

Agent implementation workflow

Successful agent implementation requires a defined, repeatable process built on the security architecture above. Each phase addresses specific security controls while mitigating the identified threats.

Phase 1: Identify requirements

Define what regulatory requirements you need to meet, what operational goals you're trying to accomplish, and what constraints you're working within. Get security, legal, compliance, and business stakeholders aligned before you build.

Phase 2: Manage supply chain risks

Supply chain integrity is a challenge across all forms of IT. When devices, services, and applications can be tampered with between source and consumer, threats can be introduced at any time. To mitigate this, component integrity must be verified and validated to be tamper-free.

Pro-tip: Don't forget, this applies to the infrastructure your code runs on too.

AI Bill of Material (AI-BOM)

The AI-BOM concept extends software composition analysis to AI components, tracking model provenance, training dataset lineage, and fine-tuning parameters. OWASP's AI-BOM extends their CycloneDX ML-BOM and is available as a **web tool**. Integrate an AI-BOM into existing supply chain security processes, treating model components with the same rigor applied to code dependencies.

If you're not running local LLMs, consider where you are getting your services. Anthropic was one of the first AI companies to achieve the **ISO 42001** certification for responsible AI.

Evaluate dependency health automatically

Most software supply chains are mostly open source, and most open-source projects have no service-level agreement. **OpenSSF Scorecard** automatically scores every dependency on signals like branch protection, fuzzing coverage, signed releases, and maintainer activity. It runs in CI and helps identify unmaintained packages. Wire it in alongside your AI-BOM so model components and code dependencies carry the same risk signals.

Audit your dependency tree for redundancy

Most large codebases accumulate multiple libraries doing the same job (several HTTP clients, several JSON parsers), each adding attack surface for no functional gain. Point a frontier model at your lockfile and ask which dependencies overlap and what migration would look like — this is typically a one-hour exercise that surfaces consolidation worth doing.

Narrow remediation with reachability analysis

Evaluate the reachability of vulnerable code so you remediate the smallest set that actually matters. Combine this with continuous delivery pipelines that run regression tests on updates, so you can deploy patches quickly with confidence you haven't broken anything.

AI vendoring for small unmaintained dependencies

For small dependencies that score poorly on Scorecards and are not actively maintained, having a frontier model reimplement the subset of functionality you actually use is often safer than continuing to depend on them. Treat this as a standard response to an unhealthy dependency, not an exotic workaround.

Cryptographic signing

Sign models and software at every stage through production deployment. Signatures verified only at deployment might not catch subsequent tampering. Runtime verification confirms ongoing integrity.

Vendor assessments

Review security practices of tool providers before adoption. Assess update mechanisms for supply chain risk and consider provider incident history and vulnerability response capabilities. Validate components at runtime to detect post-deployment tampering. Your third-party risk management process should explicitly ask suppliers how they are preparing for AI-accelerated exploit timelines and whether they are scanning their own code.

This includes free and open source software (FOSS). Download the software, assess the code directly, and evaluate the provider. Does the provider have a large community, do they have a long history of support, etc.? While this doesn't mean that another submitter couldn't insert malware, it would indicate that the original author isn't directly an adversarial actor.

Pro-tip: Run/host the MCP server yourself, on an immutable platform, after you have verified the code. Cryptographically sign it yourself, and perform the same actions on updates before introducing them to production.

Phase 3: Define agent boundaries

Define exactly what each agent is allowed to do, when it should escalate to a human for approval, and the resulting blast radius should anything go wrong.

Assign a unique identity

Every agent instance needs a unique, cryptographically rooted identifier that persists across its actions. Without a distinct identity, correlating logs during an incident becomes guesswork. You can't determine which agent accessed a resource, triggered an error, or made a specific decision. Unique identifiers enable the traceability discussed earlier, allowing you to filter audit logs by agent, reconstruct action sequences, and attribute outcomes to specific instances when investigating anomalies or breaches.

Pro-tip: Claude Code assigns a unique session.id to each session, with user.account_uuid and organization.id attribution on all telemetry events, enabling precise incident investigation without shared identity ambiguity.

Approved/prohibited actions

Document what actions are permitted or denied. Rather than leaving this implicit, write it down. An agent permitted to read customer records, summarize information, and draft responses has clear boundaries. An agent with vague permission to "help with customer service" does not.

You need to be able to implement this at a granular enforcement level. It's one thing to tell an agent "Don't do this", it's another to prohibit that action through permissions.

Pro-tip: Claude Code natively supports this level of granular access control in **settings.json**, which can be configured with global and project-level settings, and environment variables.

Escalation triggers

Escalation triggers identify what requires human review before proceeding. High-value transactions, access to sensitive data categories, or communications with external parties might all require approval. Define thresholds that balance security against operational efficiency.

Claude Code natively supports this with both **settings.json**, via the "ask" parameter, as well as **hooks**.

Scope limits / Least Agency

Scope limits constrain which systems, data, and resources the agent can access. Even within permitted actions, agents should access only the systems necessary for their function. A customer service agent doesn't need access to HR systems, even if the underlying service account technically permits it.

The best way to tackle this is to limit the access of accounts provided to agents. For example, if you are providing access to a database system via API using certificate-based authentication you would limit the access to read, unless the agent needed write access, and the read access would be limited to only the data necessary to perform its duties. Simply put, Least Agency and deny by default, at all times. If the agent were compromised or the credentials stolen, the blast radius would be severely limited.

> **Pro-tip**: Sometimes you may just want to break up some of the functions/goals of an agent into multiple agents. This compartmentalization of capabilities and access to resources means that attackers are required to compromise more agents in order to gain access to more system resources.

Very important: each agent should have a unique ID and its own access credentials. If you break it into multiple agents and provide them all the same credentials, you have failed to compartmentalize the risk.

Identify the blast radius

With the approved actions, prohibited actions, escalation triggers, and scope limits in place, identify the effective blast radius. What could go wrong if the agent or system were compromised?

Apply the "impossible vs. tedious" test here. If your containment plan relies on friction — the attacker would have to make a lot of requests, or would have to bypass several rate limits — assume it will fail. If the current risk level is still unacceptable, adjust the previous settings to further restrict what the agent is capable of doing.

Phase 4: Defend against prompt injection

Just as it's necessary to implement input sanitization on traditional technologies like databases, we need to ensure that we control and clean information that is presented to our agents. Defenses must address both direct attacks through user input and indirect attacks through external data sources.

In addition to escalation triggers and scope limitations, input isolation, constitutional classifiers, and limiting attack surfaces greatly reduce the risks of prompt injections.

Input isolation

Input isolation treats all natural-language inputs as untrusted. User-provided text, uploaded documents, and retrieved content pass through validation before influencing agent behavior. Microsoft's Spotlighting technique reduces indirect injection attack success from over **50% to under 2%** by clearly delimiting untrusted content.

Constitutional classifiers

Constitutional classifiers provide an additional detection layer. These AI-based systems scan prompts and responses for manipulation attempts. Anthropic's approach **blocked 95% of jailbreak attempts** in testing with minimal increase in over-refusal rates.

Limit attack surfaces

While a traditional security technique, reducing the attack surface is one of the most effective ways to mitigate prompt injection. Limit who or what can interact with the agentic system. If the system can be limited to trusted personnel and resources, the ability for a malicious actor to hijack your system will be severely limited.

Phase 5: Secure tool access

Tool access is one of the highest-risk surfaces in agentic deployments. When tool capabilities lack proper controls, a single compromised agent can cause widespread damage.

Tool allow-listing

Tool allow-listing restricts agents to approved tools. Rather than permitting any tool that becomes available, maintain explicit lists of permitted tools per agent function. Further, using our deny-by-default approach, reject invocations of unlisted tools.

This will take different forms depending on the agentic framework you are using — some require you to explicitly provide them to the agent in the first place, others will have them as a resource pool. Regardless of the method, you want to control this on two fronts. The first will be directly at the agent level, with implicit allow/deny permissions. The second will be outside the agent level, in case the agent or the agent environment is compromised. The easiest way to do this is to require authentication for tools: certificate-based authentication on an API interface, or short-lived tokens bound to the calling agent's identity. Static API keys are not acceptable for tool authentication, even at Foundation.

Pro-tip: Claude Code supports explicit tool based permission control at the agent level via **settings.json**, which can be configured with global and project-level settings, and environment variables.

Capability restrictions

Limit what permitted tools can do. An email tool might be restricted to reading, with send capability requiring separate authorization. A database tool might permit queries but prohibit schema changes. In larger enterprise services, such as Active Directory, this typically takes the form of role-based access controls (RBAC) on the provisioned account.

Parameter validation

Validate tool call arguments before execution. Input validation applies to tool parameters just as it does to user input. Reject parameters that exceed expected ranges or contain suspicious content.

Parameter validation can, and should, happen on both the agent side and the tool side.

Pro-tip: Claude Code natively supports this capability on the agent side through **hooks**. Using a PreToolUse hook, you can create a hook to validate the parameters before they are sent.

Sandbox execution

Sandboxed execution provides containment when tools behave unexpectedly. Container sandboxes and/or microVMs with restricted network access, limited file system mounts, and syscall filtering contain the impact of compromised tools.

Another consideration is rate limiting and spending controls to prevent resource exhaustion attacks. Where possible, implement circuit breakers that halt tool execution when usage exceeds defined thresholds, or if you have implemented Attribute-based Access Control (ABAC), where usage behavior deviates from approved baselines. Remember that rate limits are friction, not barriers: they buy time but do not stop a determined agentic attacker.

Pro-tip: Claude Code now supports **sandboxing**, with file system isolation, network isolation, and OS-level enforcement. You can delve deeper in the official **documentation**.

Approval escalation

Just like the escalation triggers discussed earlier, we apply the same controls to high-risk tool invocations, requiring them to pause for human review. Ensure you display clear descriptions of intended actions and log approval decisions. For later explainability and justification, you need to perform forensics.

Pro-tip: Claude Code supports this natively — by default all tool calls require human approval, and further granularity can be configured via **settings.json**. In addition, pre- and post-tool calling actions can be configured using **hooks**.

Phase 6: Protect agent credentials

Credential protection prevents attackers from stealing or misusing agent authentication material. When agents share credentials or operate under generic service accounts, a single compromised credential grants attackers access to every system those agents can reach. A distinct identity for each agent, backed by cryptographic authentication and hardware-rooted wherever possible, contains the blast radius of credential theft while enabling granular access control and accurate audit trails.

Static API keys, embedded credentials, and shared service-account passwords are among the first things an attacker with model-assisted code analysis will find. Treat them as already-compromised.

Short-lived, identity-provider-issued credentials as baseline

Short-lived credentials limit the window of opportunity for credential theft. Tokens that expire in minutes rather than days reduce the value of stolen credentials. Automated refresh maintains operational continuity without long-lived secrets.

Where resources permit, implement certificate-based identity with a Certificate Authority that enrolls agents, issues short-lived certificates, and maintains Certificate Revocation Lists or OCSP responders for real-time validation. For organizations without PKI expertise, cloud-native managed identity services and secret management platforms like HashiCorp Vault provide automated credential rotation and centralized revocation without the operational overhead of running a certificate authority.

Pro-tip: Claude Code natively supports **OAuth 2.0 authentication** with automatic token refresh for MCP server connections, avoiding long-lived secrets. Additionally, permissions granted during a session for tools configured as "ask" are **session-scoped** and expire when the session ends.

Hardware-bound credentials for production and sensitive workloads

For production systems and sensitive internal tools, credentials should be bound to attested hardware so that stolen credential material cannot be exported from a compromised host. This applies to calls between production services as well as to human-to-service calls. Phishing-resistant 2FA (FIDO2 or passkeys) should be the default wherever human authentication is in the loop; SMS-based codes do not meet the Foundation bar.

Credential isolation

Credential isolation ensures each agent instance has unique credentials. When agents share credentials, a single theft grants attackers the combined access of every agent using that secret, and revoking that credential disrupts all of them. Per-agent credentials contain this blast radius while enabling granular access control and accurate incident investigation. Credentials should never appear in code or configuration files; inject them at runtime from secrets management systems that log access and support emergency revocation.

Pro-tip: Claude Code stores API credentials in the OS credential store rather than configuration files. The apiKeyHelper setting can execute a script at runtime to retrieve secrets from external vaults, supporting integration with secrets management systems.

Explicit trust boundaries

Multi-agent systems require explicit trust boundaries. Agents should verify the identity and authorization of other agents before accepting delegated tasks. Implement authorization checks at each step of multi-agent workflows, rather than trusting that the initiating agent had appropriate permissions. Where possible, log all inter-agent communications and flag unusual delegation patterns for review.

Pro-tip: Claude Code spawns ephemeral sub-agents by design, which act like an extension of the original agent and can have up to the same permissions levels as it has been originally assigned. From an external observability and access standpoint, essentially there is no difference between the original agent and its sub-agents. However, the distinction is captured by Claude Code, and would be visible via OpenTelemetry or in the JSONL transcripts located in the Claude Code projects folder.

Just-in-time (JIT) access

Just-in-time access grants permissions only when needed and revokes them immediately after use. Rather than maintaining standing access, agents request credentials for specific operations, scoped to specific resources for defined durations. This approach limits exposure even if agent infrastructure is compromised: an attacker finds no cached credentials to steal. Token lifetimes should be measured in minutes rather than hours or days.

Pro-tip: JIT access is very powerful, and not easily implemented. If you can implement it within your environment, even partially, you should do so. This is considered an advanced Zero Trust implementation and a very strong threat mitigation.

Attribute-based Access Control (ABAC)

Attribute-based access control (ABAC) evaluates multiple factors before granting access, such as: agent identity, resource sensitivity, requested action, time of day, source location, and current risk score. This context-aware approach enables policies like allowing read access to low-sensitivity data while requiring step-up authentication for sensitive records, or permitting routine queries while blocking bulk exports. ABAC policies adapt to circumstances without requiring new roles for every access pattern.

Pro-tip: ABAC, like JIT, is another advanced implementation. The factors used for evaluation, and what is appropriate for each agent in its particular use case, are determined by you. When properly configured, detection and prevention of misuse is immediate.

Phase 7: Safeguard agent memory

Memory protection prevents attackers from corrupting agent context or extracting sensitive information from memory stores. Unlike attacks targeting a single session, memory poisoning persists across interactions, influencing agent behavior long after the initial compromise. Effective protection requires isolation between users and sessions, integrity verification of stored content, and policies governing how long sensitive context persists.

Memory isolation

Memory isolation enforces strict boundaries between sessions and users. Without these boundaries, poisoned context from one conversation can influence future interactions, and a compromised session can access data from previous ones. Session isolation ensures that information from one conversation cannot affect another, limiting the persistence of any successful poisoning attempt.

> **Pro-tip**: Claude Code enforces session isolation by default. Each session starts with fresh context, and sub-agents operate in their own isolated context windows without access to the parent conversation history.

Context integrity validation

Integrity checking validates persisted context before use. Cryptographic hashes detect unauthorized modification, while source attribution tracks where each memory element originated. Together, these enable organizations to identify tampering and quarantine memories derived from untrusted sources.

Implement integrity validation at every retrieval, not just at storage time. Tag each memory element with its source and the conditions under which it was added. Store hashes in tamper-resistant logs separate from the memory content itself. When validation fails, reject the suspect context and alert security teams rather than allowing potentially poisoned memories to proceed.

Context retention policies

Retention policies limit how long sensitive context persists. By applying time-to-live values and automatically expiring unverified memory, you'll prevent poisoned content from remaining active indefinitely. Shorter retention periods for high-risk context such as external inputs or unverified tool outputs reduce exposure without disrupting core operational data.

When poisoning is detected, recovery depends on preparation. Versioned memory stores enable rollback to known-good states, while quarantine procedures isolate suspected content for forensic analysis before deletion. Test rollback procedures before incidents occur, and define clear criteria for when full memory purging is warranted versus targeted remediation.

> **Pro-tip**: Claude Code supports configurable retention policies. The cleanupPeriodDays setting controls how long local transcripts persist before automatic deletion.

Additionally, checkpoints capture the state before each edit, enabling rollback to known-good states via the rewind feature (Esc+Esc or /rewind). You can restore code changes, conversation state, or both independently. For enterprise deployments, server-side retention defaults to 30 days. More information can be found at **data usage** and **checkpointing**.

Phase 8: Measure what matters

When agentic systems operate as black boxes, you cannot determine whether they are delivering intended outcomes or have been compromised and are serving attacker objectives. Visibility is critical — not just seeing what agents do, but understanding why, and receiving that information quickly enough to act. These factors determine whether teams catch divergent behavior early or face catastrophic failure.

Dwell time and coverage

Instrument dwell time (anomaly occurrence to human awareness) and coverage (fraction of alerts investigated) before anything else. These are the two metrics AI automation has the greatest leverage to move, and they matter most when exploit windows shorten.

Explainability

Decision explainability asks whether you can trace any agent action back to its triggering input and explain why the agent chose that response. For regulated industries handling financial, health, or personal data, this explainability is not optional. It enables compliance demonstration, incident investigation, and customer trust.

Security teams should be able to answer: would we know within an hour if an agent went rogue? Can the team take time off without worrying about undetected agent misbehavior? If the answers are uncertain, the foundational controls need more work.

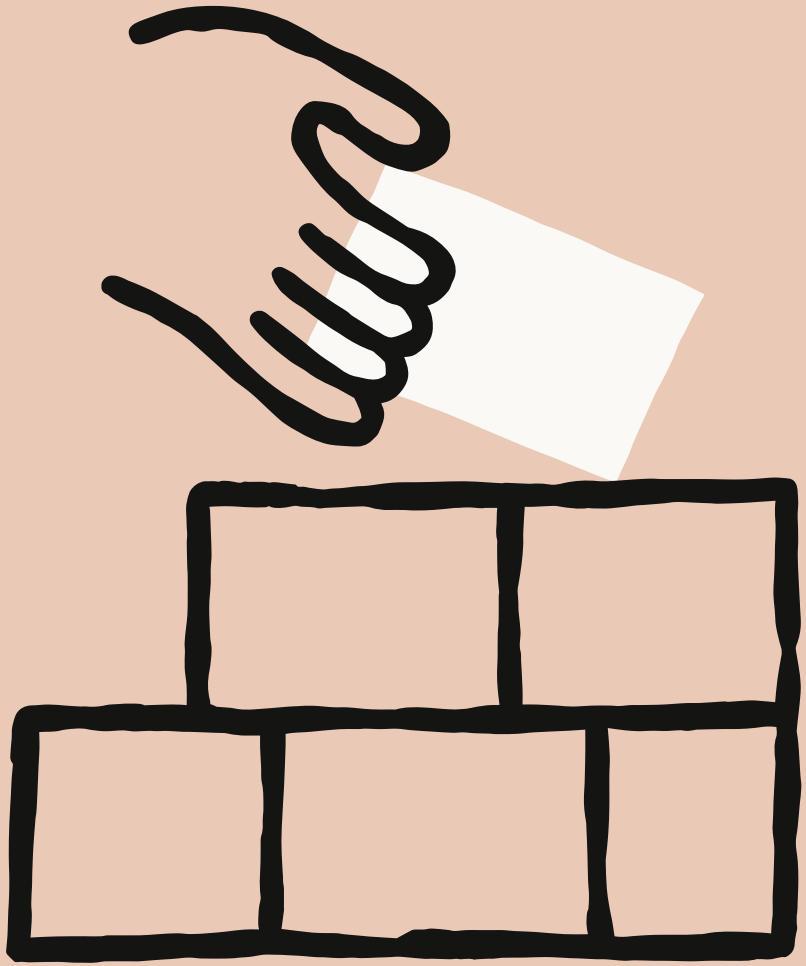
Behavior

Behavioral conformance tracks whether agent actions align with intended policies and expected patterns. Establish behavioral baselines during controlled deployment and measure drift over time. Key indicators include tool usage patterns, output characteristics, and decision distributions. An agent that suddenly favors different tools or produces outputs with changed characteristics warrants investigation, even if no single action triggers an alert.

Define acceptable variance thresholds and flag deviations for review. Continuous behavioral monitoring catches subtle compromises that evade rule-based detection, such as gradual drift from memory poisoning or slow-acting supply chain attacks.

Detection speed

Detection speed measures how quickly your team becomes aware when an agent behaves unexpectedly. The difference between minutes and days translates directly to damage contained. Target detection within an hour for critical systems. Measure from anomaly occurrence to human awareness.



Part V

Defensive operations at the speed of autonomous threats

Part V

Defensive operations at the speed of autonomous threats

Securing the agents you deploy is half the work. The other half is running security operations fast enough to contend with attackers who are themselves AI-accelerated. When exploits appear within hours of a patch, response processes that take days are too slow. Agentic adversaries might attack hundreds or thousands of systems in the time required for a human to review a single alert.

The case for autonomous defense

Traditional security operations assume humans analyze alerts and decide on responses. That methodology struggles when attackers can probe defenses, adapt techniques, and exfiltrate data faster than analysts can respond. The answer is not to remove humans from the loop — it is to move humans off the bookkeeping and onto the decisions. Automate evidence collection, enrichment, correlation, and documentation. Keep humans on containment calls, disclosure calls, and customer-comms calls. Human decision speed during an incident should never be rate-limited on evidence collection or write-ups.

Put a model at the front of your alert queue

Every inbound alert should get an automated first-pass investigation before a human sees it. A triage agent with read-only access to your SIEM and a well-scoped set of query tools can direct analyst attention to the alerts that most need human judgement.

Practical start: pick one noisy rule with a known-high false positive rate. Wire a frontier model into its alert stream with read-only access to the underlying data, and have it produce a structured disposition for every firing. Measure agreement against a human reviewer for two weeks. If the agreement rate is tolerable, expand to the next rule. Do not try to automate the whole queue at once.

Agentic security orchestration

Today, Security Orchestration, Automation, and Response (SOAR) platforms enable security teams to integrate and coordinate separate security tools, automate repetitive tasks, and streamline incident and threat response workflows.

The next generation of SOAR is Agentic SOAR, which adds adaptive capabilities that respond to novel situations. This allows flexibility beyond existing playbooks and the adaptability to directly address malicious AI-driven attacks within seconds.

Response actions for suspicious traffic or behavior could include automated quarantine or isolation at the network or system level, dynamic access control adjustments at the user or resource level, session termination, and credential revocation — all executed through the identity-based isolation and short-lived-credential infrastructure built in Part III.

Map detection coverage against MITRE ATT&CK

MITRE ATT&CK provides a standard vocabulary of attacker techniques that most detection tools already use. Knowing which techniques you can detect, and which you can't, is more useful than a general goal to "improve detection." Prioritize coverage for lateral movement and credential access. These are where AI-accelerated attackers will get the most leverage from compromised agent identities.

Atomic Red Team is an open-source library of small, safe tests mapped to ATT&CK techniques; running a handful and checking which ones your existing logging actually detected is a one-afternoon exercise that produces a concrete coverage map.

Run a tabletop for five simultaneous incidents, not one

The standard tabletop exercise assumes one critical CVE with a working exploit hits on a Monday. Run the version where five hit in the same week. Intake, triage, and remediation tracking should scale accordingly — a workflow built around a spreadsheet and a weekly meeting will not keep up. Plan for an order-of-magnitude increase in finding volume and rehearse it before it happens.

Establish emergency change procedures in advance

A two-week change-approval cycle for production patches is itself a security risk. The same applies to emergency containment actions: taking a service offline, rotating a credential, blocking a network path. Decide in advance who can authorize these, how fast they can be authorized, and what evidence is required. Practice the authorization path so it is not improvised during an incident.

Trust through verification for defensive agents

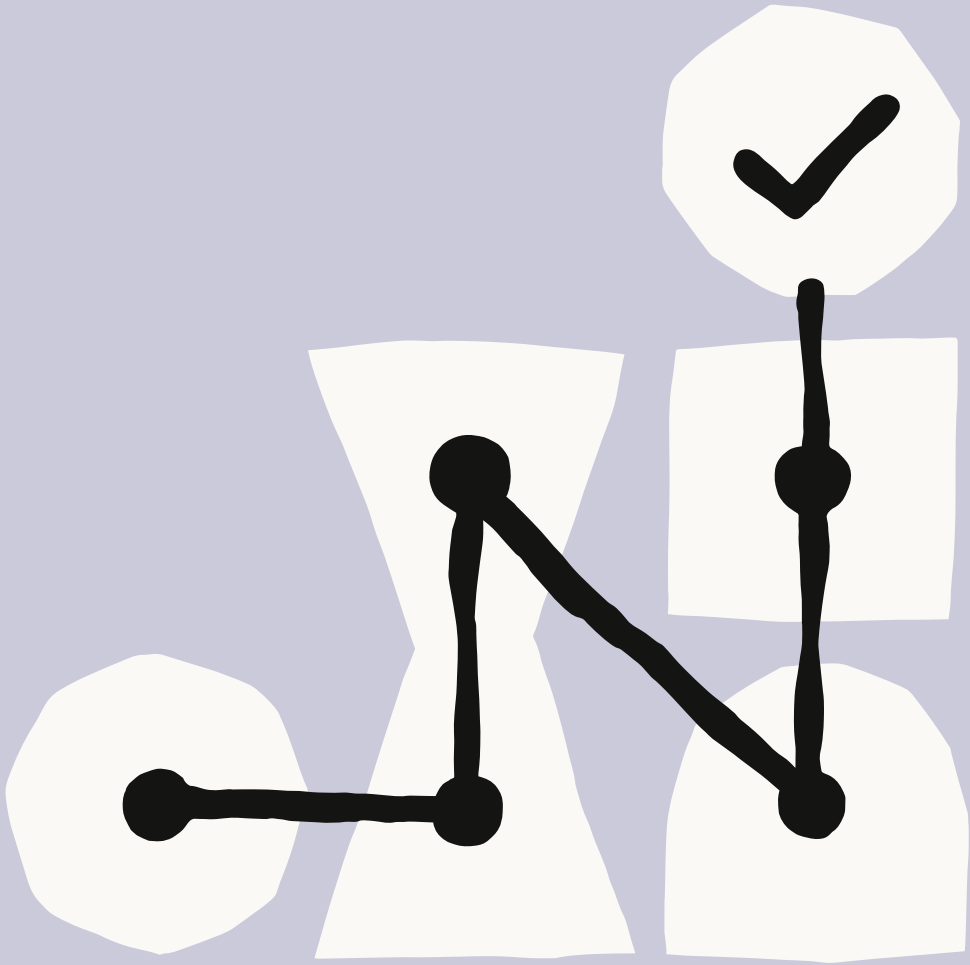
Agentic SOAR capabilities are powerful, and their blast radius can be significant. The same Zero Trust principles outlined earlier need to be applied. Organizations should not blindly trust defensive automation any more than they trust other autonomous systems.

Verified integrity ensures agentic SOAR systems have not been compromised. Attackers who compromise defensive agents gain powerful capabilities. Defensive agents should run in hardened environments with strong integrity verification.

Limited blast radius constrains what defensive agents can do. Even trusted defensive systems should operate with least privilege. Automated response capabilities should be scoped to specific actions with clear boundaries.

Clear escalation paths ensure humans remain informed and in control. Automated responses should generate alerts for human review. High-impact responses should require human approval even when automated systems recommend them.

The monitoring and audit capabilities described earlier apply to defensive agents as well. Defensive agent actions should be logged, traced, and reviewed just like any other agent activity. This ensures accountability and enables improvement of defensive capabilities over time.



Chapter 6

From principles to practice

Chapter 6

From principles to practice

Agents face unique threats, different from traditional IT. Zero Trust provides the framework to address them.

Verify every agent action, grant minimum necessary permissions, contain damage when compromise occurs. Identity enables attribution and access control. Observability reveals what happened. Behavioral monitoring detects anomalies. Input and output controls prevent attacks at boundaries. Integrity protections enable recovery. Defensive operations move at the speed of the threat.

Skip one capability and attackers exploit the gap.

The three-tier framework accommodates different organizational needs. Start at the Foundation tier — but recognize that the Foundation floor has been raised in response to AI-accelerated offense: short-lived tokens, cryptographically rooted identity, identity-based isolation, and automated first-pass triage are now entry requirements, not aspirations. Progress systematically as deployments scale and risk increases. The tiers provide a roadmap, not a finish line. Threats evolve and controls must advance with them.

For regulated industries, HIPAA, FINRA, GDPR, FedRAMP, and the EU AI Act already impose requirements that align with Zero Trust. Adoption deadlines are approaching, and competitive pressure means agent deployments aren't slowing down.

For security leaders: the compliance deadlines are real, the threat landscape is moving, and retrofitting controls after an incident costs more than building them now. The framework in this document gives your team a concrete starting point. The organizations best positioned for this shift will not necessarily be the ones with the most advanced AI. They will be the ones whose fundamentals are strong enough that AI-assisted scanning finds fewer bugs in the first place, and whose agent deployments were architected for breach from day one. For broader org-wide readiness against AI-accelerated offense, check out our blog article, **Preparing your security program for AI-accelerated offense**.

For architects and engineers: start at Foundation, validate your controls, and advance the tiers as your deployments scale. Treat the "impossible vs. tedious" test as a standing design review question. The threats will evolve. So should your defenses.

Claude

claude.ai