Agent Baseline — security outcomes for enterprise AI agents

The Enterprise Control Plane for AI Agents

A vendor-neutral reference architecture for safely building, deploying and operating AI agents, written for the CISO who has to decide about them.

Identify the agent. Bound its authority. Control its actions. Prove its outcomes.

Published as **Agent Baseline** · v1.0-draft · 30 July 2026 · agentbaseline.org Draft for public comment until 30 September 2026. Controls are cited by bare identifier — `DIS-01` — and are not yet frozen; see VERSIONING.

Executive summary

AI agents have landed on the CISO's agenda whether they were ready for them or not. CEOs and CTOs are pushing agentic tools and development across their organizations, often on aggressive timelines. Security is not always in the room when these decisions are made and may only become involved once agents already have access to sensitive data or production systems. That leaves the CISO with an uncomfortable choice: allow adoption without the right controls and take on poorly understood risk, or block it and become an obstacle to a major business priority while adoption continues in the shadows. Their job is to help the organization move quickly without losing control.

The risk is real, and the operating model is different. Unlike most enterprise software, agents can be programmed by end users at runtime through natural-language instructions, changing what they are asked to do and how they use tools or data without a conventional development and release cycle. The enterprise is therefore not just provisioning fixed-function software; it is provisioning a programmable actor. Because agents can access sensitive data, call tools and act at machine speed, a mistake or malicious instruction can become a harmful action before a person can intervene. Existing security controls still matter, but they were not built for this combination of runtime programmability, access and autonomy.

The market does not make the problem easier. CISOs looking for help quickly encounter an AI marketing fog, where claims such as "agent governance" can mean almost anything. In reality, each product solves a piece of the problem, yet it is often unclear which piece or how the products are meant to fit together. To cut through that fog, this paper defines the Agentic Control Plane by the security outcomes the enterprise must achieve: Discover, Authorize, Constrain, Validate, Observe and Respond.

The architecture extends incumbent systems rather than replacing them. Identity providers, policy engines and SIEM platforms remain enterprise systems of record and enforcement; the control plane joins them through common context, decisions and evidence. It provides a governed path from business intent to agent action and outcome.

From Business Intent to Controlled Agentic Action

Figure 1. The six required outcomes operate as one control system across enterprise intent, action, evidence and containment.

What we mean by an AI agent

An AI model is not, by itself, an agent. An AI agent is a software system that receives a goal or instruction, uses a model to determine how to pursue it, and can act through tools or connected systems with limited human supervision. Unlike most enterprise software, its effective behaviour is not fully defined before deployment. An end user can program it at runtime through natural-language instructions, changing what it is asked to do and how it uses approved tools or data.

The architecture applies to agents that can affect enterprise resources or business outcomes — for example, by accessing sensitive data, invoking an internal API or updating a system of record. It is not intended as a generic AI ethics framework, model-development standard or product comparison. Its purpose is to operationalize and enforce security, governance and risk decisions around agents.

An agent execution begins with business intent from a person, service or business event and turns it into enterprise actions that affect protected resources and business outcomes. The Agentic Control Plane does not prescribe how the model reasons; it governs the conditions under which the agent and each material action may proceed.

- Discover: Establish the operating envelope.
- Authorize, Constrain and Validate: Control each material action.
- Observe: Creates the correlated evidence needed to understand and prove what happened.
- Respond: Provides stop, revocation, quarantine and impact-scoping capabilities.

No single enforcement point can deliver this control system. The outcomes are logically unified but physically distributed across identity, data, runtime, tool, application, transaction, observability and evidence boundaries. Common context and control identifiers allow those systems to operate as one path from intent to outcome.

The six outcomes are the minimum category test for an Agentic Control Plane. A product may implement one or more components, but end-to-end control requires all six outcomes to be addressed through an integrated architecture.

OUTCOME	REQUIREMENT	EVIDENCE OF ACHIEVEMENT
1. Discover	Discover every agent and its dependencies.	Authoritative inventory reconciled with observed code, cloud, endpoint, identity, SaaS and runtime evidence.
2. Authorize	Authorize the agent's purpose, identity and delegated authority.	Attributable identity and a purpose-, task-, resource-, action- and time-bound delegation context.
3. Constrain	Constrain runtime, tools, data and permissible actions.	Capability profile, tool and data allowlists, scoped credentials, transaction limits and approval decisions.
4. Validate	Validate composition, behavior and outcomes.	Approved system manifest, security evaluations, drift checks and independent outcome validation.
5. Observe	Observe every material decision, action and dependency.	Correlated telemetry that joins intent, actor, task, policy, action, target, outcome and cost.
6. Respond	Stop, revoke, quarantine and contain unsafe agent activity.	Tested circuit breakers, revocation, quarantine, escalation, evidence preservation and impact scoping.

Category test. The six outcomes are a category test, not a certification scheme. A product may cover one or more; an enterprise architecture is complete only when all six are addressed. A control plane must do more than inventory or observe agents. It must preserve policy and evidence from approved purpose through execution and outcome, and it must be able to stop, contain and investigate unsafe activity.

Six-outcome Agentic Control Plane

The six outcomes are not an exhaustive list of controls. They are the minimum category test because together they answer three questions any complete Agentic Control Plane must address: What is operating and under whose authority? Is it staying within approved boundaries? Can the enterprise prove what happened and stop it when necessary?

Discover, Authorize, Constrain, Validate, Observe and Respond cover that full path from business intent to action, evidence and containment. A product may implement one or more components, but end-to-end control requires all six outcomes to be addressed through an integrated architecture.

Discover

AI agents can be created and deployed with little infrastructure or specialist support. Developers can build them with code frameworks; employees can assemble them through low-code platforms or adopt public services without involving security or procurement. These agents may not appear in a

traditional software inventory, yet they may receive sensitive data, connect to internal systems or act using employee credentials.

The same visibility problem applies to components. Third-party models, MCP servers, skills and plugins can be introduced without review, creating supply-chain risk. Internally developed components can be just as consequential: a skill capable of deploying to production may allow an agent to make destructive changes.

Finding an agent is not enough. Security must know what it can access and do, who is accountable for it, and the business context in which it operates. Evidence is fragmented across cloud environments, source repositories, endpoints, identity systems and SaaS platforms. The required view changes as integrations are added and agents move into production, so registration must be supported by continuous discovery and reconciliation.

Control requirements: DIS-01 Authoritative agent registry; DIS-02 Authoritative component registry; DIS-03 Ownership and risk context; DIS-04 Status and decision history; DIS-05 Automated discovery and reconciliation; DIS-06 Agent composition mapping; DIS-07 Effective-access mapping. (Full text in Appendix A.)

Authorize

Identity and attribution. When an agent acts against an enterprise system, the organization must determine which agent, which deployed instance and which initiating principal or approved autonomous purpose the action belongs to. Shared service credentials cannot provide this attribution. They collapse many users, agents and tasks into one technical identity and prevent reliable reconstruction when something goes wrong.

Every party to an action therefore carries a distinct, verifiable identity. The agent authenticates as itself; the initiating human or service authenticates as itself; and each downstream agent retains its own identity. These identities are joined for the specific action, task and time through a delegation context. One agent serving many users is associated with the principal for the current run, while a service- or event-initiated workflow resolves to a named autonomous purpose and accountable owner. The result is an attributable action without pretending that the agent and principal are the same actor.

Verification rests on identity and attested run context, not on a reusable credential stored by the agent. Existing identity providers remain the source of truth. The control plane adds the agent, deployment, run, task and delegation context that conventional authentication events do not capture.

Identity and delegation — A downstream agent authenticates as itself while carrying the originating principal and recorded delegation path. Identity answers who acted; delegation answers on whose behalf and with what authority.

Delegated authorization and policy. When an agent acts incorrectly, the damage should remain within the task it was authorized to perform. In many deployments, broad standing access persists between actions, so the scale of failure is set by the credentials the agent holds rather than by the work in front of it.

The required property is per-action least privilege. A proposed action against a target resource enters an authorization boundary; a decision to allow, restrict, require approval, quarantine or halt leaves it. The decision binds the initiating principal or approved autonomous purpose, agent and run, task, target resource, requested action, limits and governing policy. A short-lived credential or action permit is issued only after that decision and is not retained by the agent.

Multi-agent delegation preserves both identities and authority. Each hop records the calling agent, receiving agent and authority passed, while the originating principal or autonomous purpose remains unchanged. The receiving agent may receive no more authority than the delegating agent holds, and policy can narrow it further. Circuit breaking is part of the same boundary: the point that permits an action must also be able to deny or halt it when context or behavior changes.

Control requirements: AUT-01 Distinct identity and action attribution; AUT-02 Purpose- and task-bound authority; AUT-03 Delegation attenuation; AUT-04 Just-in-time credentialing; AUT-05 Independent approval; AUT-06 Fail-closed authorization and circuit breaking.

Constrain

Composition and supply-chain enforcement. An agent's supply chain is wider than a traditional application's. Alongside images, packages and libraries, it may rely on models, datasets, prompts, tools, MCP servers, skills and plugins. Any of these can introduce vulnerabilities, malware, tampering or compliance risk.

A component may also be unsafe because of how it is configured. A tool or MCP server can receive broader permissions, data access or system privileges than its function requires, increasing the impact if it fails or is compromised. Sensitive data may also cross into external or lower-trust components without appropriate boundaries.

Risk can emerge from the combination of components, even when each appears benign on its own. The "lethal trifecta," for example, combines access to private data, exposure to untrusted content and external communication, creating a route from prompt injection to data theft. A similar toxic flow can connect untrusted input to a destructive tool.

Discovery establishes which components exist and where they are used. Supply-chain and composition controls determine which components may be used, how they may be connected, and whether agents can introduce unapproved components at runtime.

Control requirements: CON-01 Agentic system manifest; CON-02 Approved and verified components; CON-03 Toxic capability combinations; CON-04 Component-scoped authority; CON-05 Rug-pull protection.

Runtime isolation. An agent does not merely reason. It runs code, calls tools and starts processes. When this occurs inside a developer environment or general-purpose CI runner, the agent may inherit the host's filesystem, credentials, network and unrelated projects. An incorrect instruction or malicious content then executes with the reach of the host rather than the reach of the task.

Each agent task should run inside a small environment whose capability is limited to the work in front of it. The agent sees only an approved workspace, reaches only approved network destinations, receives bounded compute and duration, and cannot access host credentials or unrelated workloads.

The environment is created from an approved template, can be paused or snapshotted when required, and is reset or destroyed at completion. Policy is centrally defined and locally enforced so it cannot be disabled by the agent or an ordinary user.

Sandbox operating model. The boundary must remain usable or teams will bypass it. A practical operating model uses warm pools and approved templates to claim an environment quickly, attach the required workspace and policy, preserve a stable run identity, snapshot when necessary and destroy the environment afterward. The same lifecycle supports oversight: enforcement occurs at the boundary, a circuit breaker can halt a run, material actions can be sent for approval, and events can be correlated with the initiating user and task. Local developer sandboxes and remote cluster environments can implement the same policy contract while using different isolation technologies.

Control requirements: CON-06 Isolated execution; CON-07 Filesystem, network and resource confinement.

Validate

Admission and continuous reassessment. Existing secure-development controls do not cover everything that determines how an agent behaves. Prompts, skills, tool definitions, memory configuration and agent policies often sit outside conventional code review and testing. An agent can therefore pass ordinary application-security checks and still be vulnerable to prompt injection or tool misuse.

The same problem applies to third-party agents and components, where the organization may have limited visibility into how they were developed, how they handle data or how they change over time. Any approval is therefore a point-in-time judgement: a change to the model, tools, permissions or components can materially alter the risk without changing the application code.

Control requirements: VAL-01 Production admission gate; VAL-02 Agent-specific security validation; VAL-03 First-party component assurance; VAL-04 Third-party agent assurance; VAL-05 Third-party component assurance; VAL-06 Revalidation.

Output validation. Coding agents can create or modify code, configuration and dependencies at machine speed. Without the same security and release checks applied to human-produced changes, they can introduce vulnerable or malicious artifacts into production.

Control requirements: VAL-07 Agent-generated artifact assurance.

Observe

Create protected, correlated telemetry and an investigable record that explains what the agent intended, requested, was permitted to do, actually did and caused.

Agent-native observability. Agent behavior is nondeterministic and can change as models, tools and operating context evolve. Preventive controls reduce risk but do not eliminate it. Malicious input, user error or agent failure can cause harmful actions within permitted boundaries. Whether an action is acceptable depends not only on authorization but on the task and circumstances in which it occurs.

Traditional observability provides only part of this context. A downstream system may record an API call by a service account without showing the user, agent, task, model, tool or policy decision that caused it. Agent telemetry must connect the actor, task, action, target, time, decision and outcome across systems. Operational signals such as token use, tool calls, latency, resource consumption and cost are also relevant because abnormal patterns may indicate a runaway, malfunctioning or compromised agent.

Richer telemetry creates its own exposure. Prompts, responses and tool results may contain credentials, personal data or corporate secrets. Observability must therefore collect the minimum content required for investigation, apply redaction and access control, and align retention with the sensitivity of the evidence.

Accountability and evidence. After an agentic workflow completes, the organization must reconstruct its intent, decisions, actions, approvals and outcome to a standard sufficient for investigation, audit or regulatory review. In many current deployments, the required events are scattered, briefly retained or tied to shared identities that cannot show who acted. A workflow that cannot be accounted for does not meet the control objective even if it completed successfully.

Accountability is an investigable record: a chain of trust assembled from the evidence created by identity, authorization, runtime, tools, applications and outcome controls. A specific workflow goes in; a defensible account of what occurred comes out. The record shows who or what initiated the work, which agents participated, how authority passed between them, which policy governed each material action, which resources were affected, which approvals were obtained and what outcome resulted.

Tamper evidence and signatures can strengthen this record, but they are not sufficient. A signature can show that a record has not changed; it does not by itself show that the record is complete, explain the sequence, or make a harmful action accountable. The essential property is investigability: enough correlated, trustworthy evidence to reconstruct the path with a workable degree of confidence.

The record feeds existing SIEM, audit ledger and GRC platforms rather than replacing them. No single product should be assumed to assemble a complete account across identity, runtime, supply chain and business systems. The architectural requirement is an evidence contract, correlation service and evidence store, with remaining manual gaps made explicit.

Observability and accountability — Observability emits and correlates runtime signals. Accountability consumes those signals to produce a workflow-level account that can be investigated, challenged and evidenced after the fact.

Control requirements: OBS-01 Agent-native telemetry; OBS-02 End-to-end correlation; OBS-03 Behaviour and drift monitoring; OBS-04 Permitted-action harm detection; OBS-05 Intent-to-outcome evidence; OBS-06 Evidence integrity, completeness and protection.

Respond

Stop unsafe activity quickly, revoke active authority, contain affected versions and components, preserve evidence and determine the scope of impact.

Response is deliberately limited to containment in this architecture. Restoring business state remains a workflow-specific operational responsibility outside the current scope.

Control requirements: RES-01 Immediate stop and authority revocation; RES-02 Version and component quarantine; RES-03 Impact scoping and evidence preservation; RES-04 Safe failure and non-agent fallback.

Conclusion

Enterprise agents change the object of security from a model or software pipeline to a dynamic action system. The central risk is not only an incorrect response; it is the transformation of intent into consequential action through changing components, identities, tools, data and trust boundaries.

The Agentic Control Plane governs that transformation. Its six outcomes are the minimum an enterprise architecture must address: discover every agent, assign ownership, authorize purpose and delegated authority, constrain capability, validate the system and outcome, observe material activity, and respond through stop, revocation, quarantine and impact scoping.

The control plane is not a new agent platform and is not a single mandatory proxy. It is the security and governance layer that allows enterprises to use many agent platforms safely. It remains modular by design, but common identity, policy, action and evidence contracts make the modules operate as one control system from business intent to business outcome.

Identify the agent. Bound its authority. Control its actions. Prove its outcomes.

Appendix A. Canonical controls

(Canonical naming + requirement set; grouped by primary outcome. Publication-ready release extends each entry with lifecycle stage, assurance profile, accountable actor, policy decision point, enforcement point, evidence, test method, failure behavior, exception handling and framework mappings.)

Discover: DIS-01 Authoritative agent registry · DIS-02 Authoritative component registry · DIS-03 Ownership and risk context · DIS-04 Status and decision history · DIS-05 Automated discovery and reconciliation · DIS-06 Agent composition mapping · DIS-07 Effective-access mapping.

Authorize: AUT-01 Distinct identity and action attribution · AUT-02 Purpose- and task-bound authority · AUT-03 Delegation attenuation · AUT-04 Just-in-time credentialing · AUT-05 Independent approval · AUT-06 Fail-closed authorization and circuit breaking.

Constrain: CON-01 Agentic System Manifest · CON-02 Toxic-flow analysis · CON-03 Component least privilege and trust boundaries · CON-04 Isolated execution · CON-05 Filesystem, network and resource confinement · CON-06 Approved and verified components · CON-07 Runtime component and generated-artifact policy.

Validate: VAL-01 Deployment admission · VAL-02 Agent-specific security testing · VAL-03 Third-party assurance · VAL-04 Independent outcome validation · VAL-05 Material-change revalidation.

Observe: OBS-01 Agent-native telemetry · OBS-02 End-to-end correlation and behavior monitoring · OBS-03 Intent-to-outcome evidence · OBS-04 Evidence integrity, completeness and protection.

Respond: RES-01 Immediate stop and authority revocation · RES-02 Version and component quarantine · RES-03 Impact scoping and evidence preservation · RES-04 Safe failure and non-agent fallback.
