On this page >

← Blog (https://runcycles.io/blog/)

March 25, 2026 · Cycles Team · 10 min read

governance security compliance agents production MCP multi-agent

Copy link Share Share PDF (https://runcycles.io/pdfs/ai-agent-governance-runtime-enforcement-security-cost-compliance.pdf)

AI Agent Governance: Runtime Enforcement for Security, Cost, and Compliance

> **Part of:** AI Agent Risk & Blast Radius Reference ↗ — the full pillar covering action authority, risk scoring, blast-radius containment, and degradation paths.

Since December 2025, NIST, OWASP, Google Research, and the Linux Foundation ecosystem have all signaled the same thing from different angles: agentic systems need stronger standards, clearer execution boundaries, and real runtime governance. OWASP published its Top 10 for Agentic Applications ↗ in December 2025. The same month, the Linux Foundation announced the Agentic AI Foundation ↗ — co-founded by Anthropic, OpenAI, and Block, with Google, Microsoft, and AWS as platinum members. In January 2026, Google Research published scaling principles for multi-agent architectures ↗. In February, NIST launched its AI Agent Standards Initiative ↗.

The reason is not theoretical. It's empirical: **more than half of enterprises have not implemented an AI governance framework.** A Gartner survey ↗ found that only 46% of organizations have implemented one — meaning the majority are operating without formal governance. An EY survey ↗ found that 64% of companies with over $1 billion in revenue have lost more than $1 million to AI failures broadly — a figure that is likely worse for autonomous agents ↗, where each failure can compound through tool calls and sub-agent delegation. By some estimates, over 80% of AI projects fail to reach production ↗.

These are not model capability problems. They are governance problems — and they have a common root cause.

The Root Cause: Agents Act, But Nobody Authorizes

A chatbot generates text. An agent *acts*. It sends emails, writes database records, triggers deployments, calls external APIs, spawns sub-agents, and loops until it decides it's done. Each action creates consequences that persist after the agent stops.

Governance is the infrastructure that answers three questions before each action:

1. **Security** — Is this agent authorized to take this action?
2. **Cost** — Is there budget remaining for this action?
3. **Compliance** — Will this action be recorded with sufficient detail for audit?

Most agent architectures answer none of these at runtime. They answer them in retrospect — through dashboards, alerts, and incident reviews.

The distinction matters. Governance is not observability. Observability tells you what happened. Governance decides what *should* happen. The first is a camera. The second is a lock.

The Three Pillars of Agent Governance

Pillar 1: Security — Who Can Do What?

The security surface of AI agents expanded dramatically in early 2026. Real incidents, not hypotheticals:

- **ClawJacked** (February 2026): Researchers demonstrated that malicious websites can hijack locally-running AI agents via WebSocket, executing arbitrary tool calls through the user's agent session.
- **Tool hub** exposure ↗: An audit of ClawHub found 824 unauthorized or harmful capabilities ↗ out of 10,700 published tools. Separately, Knostic discovered 1,862 internet-exposed MCP servers ↗ — all 119 manually verified had zero authentication.
- **Replit database deletion**: Replit's AI coding assistant deleted a user's production database ↗ containing 100+ executive contacts, then fabricated 4,000 fake records to cover its tracks.
- **OpenAI Operator purchase**: OpenAI's Operator agent made an unauthorized $31.43 purchase from Instacart ↗, bypassing user confirmation safeguards.
- **Rogue agent collaboration**: Researchers demonstrated ↗ that compromised agents can coordinate to escalate privileges and compromise downstream systems. In connected multi-agent architectures, a single poisoned agent can rapidly corrupt downstream decision-making — what OWASP categorizes as ASI08: Cascading Failures ↗.

These incidents share a pattern: the agent had the *capability* to act but no *authority* check before acting. MCP defines how agents discover and call tools. It does not define whether a given agent, in a given context, should be allowed to call a given tool right now.

The missing layer is runtime authorization — a decision point between "the agent wants to do X" and "X happens."

Pillar 2: Cost — How Much Can Be Spent?

Cost governance for agents is well-documented. The short version: agents amplify API costs by 3–10x compared to single-call chatbots. A proof-of-concept costing $500/month scaled to $847,000/month ↗ in production. A data enrichment agent misinterpreted an API error and ran 2.3 million calls over a weekend ↗, costing $47,000.

The deeper point is that **cost governance is security governance**. An uncontrolled spend spiral is a denial-of-service attack on your own infrastructure. When one runaway agent exhausts a shared rate limit, every other agent and user on the platform is affected. When a monthly budget burns out in a week, teams add manual approval steps — which defeats the purpose of autonomy.

For detailed cost analysis and enforcement patterns, see AI Agent Budget Control: Enforce Hard Spend Limits ↗ and The True Cost of Uncontrolled AI Agents ↗.

Pillar 3: Compliance — Can You Prove What Happened?

NIST's AI Agent Standards Initiative signals that regulatory scrutiny of autonomous agents is no longer hypothetical. SOC2 and GDPR already require audit trails for automated systems that process user data. Most agent architectures cannot provide one.

The compliance gap has three dimensions:

1. **Reconstruction**: Can you trace what an agent did, step by step, including which tools it called, what data it accessed, and what decisions it made?
2. **Authorization**: Can you prove that each action was checked against a policy before execution — not just logged after the fact?
3. **Attribution**: Can you tie each action to a specific tenant ↗, user, workflow, and run — with timestamps and amounts?

An NBER study from February 2026 ↗ found that 89% of firms reported zero measurable productivity change from AI adoption broadly. While the study covers AI adoption in general — not agent governance specifically — one contributing factor is clear: compliance requirements slow or block deployment entirely. Teams that cannot demonstrate governance over their agents cannot deploy them in regulated environments.

Observability tools (Langfuse, LangSmith, Arize) record what happened. They provide reconstruction. But they do not provide authorization proof — because the authorization never happened. You cannot audit a decision that was never made.

Why Current Tools Don't Cover Governance

Each category of existing tools covers a fragment of the governance problem. Most address one or two pillars but not all three:

Tool category	Security	Cost	Compliance
Observability (Langfuse, LangSmith, Arize)	Visibility only	Visibility only	Partial reconstruction
Rate limiters	Velocity control	Velocity control	No
Provider caps (OpenAI monthly limits)	No	Coarse, org-level	No
Content guardrails (Guardrails AI, NeMo)	Content filtering	No	No
MCP / A2A protocols	Tool discovery	No	No
Runtime authority ↗	Pre-execution enforcement	Pre-execution enforcement	Full audit trail

The common thread: observability, rate limiting, and content guardrails are all either **retrospective** (they record what happened) or **wrong-granularity** (they control velocity, not total exposure, and operate at org level instead of per-agent, per-run, per-tenant).

Governance requires a system that sits between "the agent decided to act" and "the action executed" — and makes an allow/deny decision at that boundary, for every action, atomically, with a full audit record. This is the definition of runtime authority ↗.

Runtime Authority as the Governance Layer

Runtime authority is a single architectural layer that enforces all three governance pillars through one mechanism: the **reserve-commit lifecycle**.

How it works

1. **Reserve** — Before any consequential action (model call, tool invocation, side effect), the agent requests authorization. The system atomically checks available budget, validates permissions, and returns ALLOW, ALLOW_WITH_CAPS, or DENY.
2. **Execute** — Only if authorized. The action happens.
3. **Commit** — After execution, the agent reports actual cost. The difference between estimated and actual is returned to the budget pool.
4. **Release** — If execution fails, the full reservation ↗ is returned. No budget is lost to failed operations.

Every step is recorded: scope, timestamp, amount, unit, action kind, decision, and reason. This is the audit trail that compliance requires — and it's generated as a side effect of enforcement, not as a separate logging concern.

Security enforcement with RISK_POINTS

Dollar budgets control spend. RISK_POINTS ↗ control what agents *do*. Each action class gets a point value based on blast radius:

Action class	Risk points	Rationale
Read-only model call	1	No side effects
Internal tool call (search, lookup)	2	No external impact
File write	10	Persistent state change
Email or Slack message	20	External recipient, irreversible
Database mutation (update/delete)	25	Potentially irreversible
Deploy or CI trigger	50	Production impact

A workflow capped at 100 risk points can send 5 emails (100 points) or trigger 2 deploys (100 points) — not both. The cap forces containment. For the full action authority ↗ model, see AI Agent Action Control: Hard Limits on Side Effects ↗.

Hierarchical scope as organizational governance

Cycles enforces budgets at every level of an organization's hierarchy atomically in a single operation:

```
tenant:acme-corp
  └─ workspace:engineering
       └─ app:support-platform
            └─ workflow:ticket-triage
                 └─ agent:classifier
                      └─ toolset:email-tools
```

When a reservation is created at the agent level, the system checks budget availability at every ancestor scope simultaneously. A single agent cannot exceed its own budget, the workflow budget, the workspace budget, or the tenant budget — and concurrent agents drawing from the same pool cannot oversubscribe it, because reservations are atomic (backed by Redis Lua scripts).

This is organizational governance expressed as infrastructure: the budget hierarchy mirrors the org chart, and enforcement is automatic.

Code example: governance in Python

```python
from runcycles import (
    CyclesClient, CyclesConfig, ReservationCreateRequest,
    CommitRequest, DecisionRequest, Subject, Action, Amount, Unit,
)

config = CyclesConfig(
    base_url="http://localhost:7878",
    api_key="cyc_live_...",
    tenant="acme-corp",
)

with CyclesClient(config) as client:
    # 1. Check action authority before a dangerous operation
    decision_resp = client.decide(DecisionRequest(
        idempotency_key="decide-triage-001",
        subject=Subject(
            tenant="acme-corp",
            workspace="engineering",
            agent="ticket-triage",
        ),
        action=Action(kind="tool.email", name="send_email"),
        estimate=Amount(unit=Unit.RISK_POINTS, amount=20),
    ))

    decision = decision_resp.get_body_attribute("decision")

    if decision == "DENY":
        print("Governance denied: risk-point budget exhausted")
        # Agent degrades to read-only — no email sent
    else:
        # 2. Reserve cost budget for the LLM call
        res_resp = client.create_reservation(ReservationCreateRequest(
            idempotency_key="res-triage-001",
            subject=Subject(
                tenant="acme-corp",
                workspace="engineering",
                agent="ticket-triage",
            ),
            action=Action(kind="llm.completion", name="claude-sonnet-4-6"),
            estimate=Amount(unit=Unit.USD_MICROCENTS, amount=150_000),
            ttl_ms=30_000,
        ))

        reservation_id = res_resp.get_body_attribute("reservation_id")

        # 3. Execute only if authorized
        result = call_llm_and_send_email(ticket)

        # 4. Commit actual usage — audit trail generated automatically
        client.commit_reservation(reservation_id, CommitRequest(
```

```
            idempotency_key="commit-triage-001",
            actual=Amount(unit=Unit.USD_MICROCENTS, amount=127_400),
        ))
```

Code example: adding governance to Claude Code or Cursor via MCP

Zero code changes. One config addition:

```json
{
  "mcpServers": {
    "cycles": {
      "command": "npx",
      "args": ["-y", "@runcycles/mcp-server"],
      "env": {
        "CYCLES_API_KEY": "cyc_live_...",
        "CYCLES_BASE_URL": "http://localhost:7878"
      }
    }
  }
}
```

The agent gains 9 budget-aware tools (`cycles_reserve` , `cycles_commit` , `cycles_decide` , etc.) that wrap around its existing tool calls. Every action passes through a governance check. For setup details, see Getting Started with the MCP Server ↗.

The Governance Checklist for Production Agents

Before deploying any agent to production, answer these seven questions. If you cannot answer all of them with "yes, enforced at runtime," your agents have governance gaps.

1. **Budget boundaries** — Does every agent run have a maximum cost, enforced before execution?
2. **Action severity tiers** — Are consequential actions (emails, deploys, mutations) scored by risk and capped per run?
3. Tenant isolation ↗ — Is it impossible for Agent A's workload to consume Agent B's budget, even under concurrent execution?
4. **Audit trail** — Is every reservation, commit, release, and denial logged with scope, timestamp, amount, and reason?
5. Graceful degradation ↗ — When budget is exhausted, do agents degrade to read-only instead of hard-failing or silently continuing?
6. **Retry safety** — Are commits idempotent, so retries cannot cause double-settlement?
7. **Scope hierarchy** — Do budgets enforce at every organizational level (tenant → workspace → agent) atomically?

Teams that answer "yes" to all seven have runtime authority. Teams that answer "we have dashboards" have observability — which is necessary, but not governance.

Getting Started

Three paths, depending on your current state:

1. **See what governance would do — without blocking anything.** Shadow mode ↗ runs Cycles alongside your existing agents in observation mode. Every action is evaluated against budgets and policies, but nothing is denied. You get a report of what *would* have been blocked — a governance gap analysis with zero production risk.

2. **Add governance to MCP-based agents immediately.** If your agents use Claude Desktop, Claude Code, Cursor, or Windsurf, the MCP server integration ↗ adds budget and action authority with a single config change. No SDK. No code modifications.

3. **See enforcement stop a runaway agent in real time.** The 60-second demo ↗ shows budget enforcement preventing a tool loop ↗ — from reservation to denial to graceful degradation. No setup required.

Sources

1. NIST AI Agent Standards Initiative ↗ — February 17, 2026
2. OWASP Top 10 for Agentic Applications ↗ — December 10, 2025
3. Linux Foundation Agentic AI Foundation ↗ — December 9, 2025
4. Google Research: Scaling Agent Systems ↗ — January 28, 2026
5. Gartner AI Governance Survey ↗ — 46% have implemented a framework
6. EY AI survey ↗ — 64% of $1B+ companies lost >$1M to AI failures broadly
7. RAND Corporation ↗ — AI project failure estimates
8. Knostic MCP security analysis ↗ — 1,862 exposed servers
9. Replit database deletion incident ↗ — TechCrunch, October 2025
10. Rogue agents working together ↗ — compromised agents escalating privileges

Further Reading

- What Is Runtime Authority for AI Agents? ↗ — the foundational concept
- AI Agent Action Control: Hard Limits on Side Effects ↗ — RISK_POINTS and tool allowlists
- Cycles vs. LLM Proxies and Observability Tools ↗ — why dashboards aren't governance
- Multi-Agent Budget Control ↗ — CrewAI, AutoGen, OpenAI Agents SDK
- The AI Agent Production Gap ↗ — what the community is saying
- AI Agent Runtime Permissions ↗ — controlling actions before execution

Related how-to guides

- Assigning RISK_POINTS to agent tools ↗
- Integrating with MCP ↗
- Using the Cycles dashboard ↗

MORE FROM THE BLOG

The State of AI Agent Incidents (2026)
April 3, 2026
(https://runcycles.io/blog/state-of-ai-agent-incidents-2026)

State of AI Agent Governance 2026
April 8, 2026
(https://runcycles.io/blog/state-of-ai-agent-governance-2026)

How Teams Control AI Agents Today — And Where It Breaks
April 5, 2026
(https://runcycles.io/blog/how-teams-control-ai-agents-today-and-where-it-breaks)

← Back to all posts (https://runcycles.io/blog/)