On this page >

← Blog (https://runcycles.io/blog/)

April 3, 2026 · Albert Mavashev · 13 min read

incidents governance security costs agents production MCP OWASP multi-agent

Copy link Share Share PDF (https://runcycles.io/pdfs/state-of-ai-agent-incidents-2026.pdf)

The State of AI Agent Incidents (2026): Failures, Costs, and What Would Have Prevented Them

AI agents are shipping to production faster than the infrastructure to control them. The result is a growing catalogue of incidents — runaway costs, wrong actions, security exploits, and cascading multi-agent failures — that share a common root cause: **no pre-execution enforcement**.

This report catalogues documented incidents and recurring failure patterns, scores each by cost and blast radius, and maps them to the runtime controls that would have prevented them.

Key findings

- **20+ documented incidents and recurring patterns** across cost, action, security, and multi-agent categories
- **Costs in this report range from $1.40 to $12,400 per incident** in direct model spend (documented and pattern-based), with business impact reaching $50,000+ from a single $1.40 agent run
- **Some of the most damaging incidents cost very little in** tokens ↗. A $1.40 model run caused $50K+ in pipeline damage ↗. A $0.80 run triggered an unauthorized purchase ↗. A $2.00 run deleted a production database ↗. Dollar budgets alone cannot prevent the worst failures.
- **Up to 84.2% attack success rate** for tool poisoning in benchmark settings under auto-approval (MCP-ITP ↗)
- **41–87% failure rates** in multi-agent coordination (UC Berkeley MAST study ↗)
- **64% of $1B+ companies** have already lost >$1M to AI failures broadly (EY survey ↗)

How to read this report

Each incident includes:

- **What happened** — the failure, in one paragraph
- **Cost** — model spend vs business impact (where both are known)
- **Source** — linked to the original disclosure, research paper, or reporting
- **Root cause** — why existing controls didn't prevent it
- **Prevention** — which runtime control would have stopped it before execution

Incidents are categorized as:

- **Documented** — sourced from public disclosures, research papers, vendor post-mortems, or security advisories
- **Pattern-based** — constructed from real failure modes observed across production deployments (marked with ⚙️)

Category A: Cost Explosions

Agents that spend more than expected — through loops, retries, fan-out ↗, or scope creep. These are pattern-based scenarios (⚙️) constructed from real failure modes — see Categories B and C for externally documented incidents from named companies and security researchers.

A1. Coding agent retry loop — $4,200 ⚙️

A coding agent hit an ambiguous error, retried with expanding context windows, and looped 240 times over three hours ↗. Total cost: $4,200. Three dashboards showed the spend in real time. None could stop it.

	Detail
Model cost	$4,200
Business impact	Budget exhausted, all agents blocked by provider cap
Root cause	Provider cap is monthly/org-wide — doesn't enforce per-run
Prevention	**Budget gate** — $15 per-run cap stops at 8 iterations

A2. Weekend backlog processing — $12,400 ⚙️

A coding agent deployed Friday afternoon processed a 2,300-item backlog over the weekend ↗ without budget enforcement. Context windows grew per item, retries compounded, and nobody checked until Monday.

	Detail
Model cost	$12,400
Business impact	Weekend budget consumed, Monday recovery
Root cause	No per-batch or per-task budget limit
Prevention	**Budget gate** — per-task cap of $5 limits total to ~$2,500

A3. Concurrent agent burst — 6.4x overrun ⚙️

Twenty concurrent agents processing 200 documents simultaneously ↗ hit a TOCTOU race condition. All read "budget remaining: $500" and all proceeded. Actual spend: $3,200.

	Detail
Model cost	$3,200 (budget was $500)
Business impact	6.4x budget overrun
Root cause	Application-level counter lacks atomicity
Prevention	**Atomic** reservation ↗ — budget locked before execution, concurrent reads see accurate remaining

A4. Retry storm during CRM outage — $1,800 ⚙️

A CRM returns 500 errors for 12 minutes. Retry logic at tool, step, and orchestration layers compound ↗ — 27x multiplication across 45 active conversations. Cost: $1,800 in 12 minutes.

	Detail
Model cost	$1,800
Business impact	All tenant ↗ budgets affected during the storm
Root cause	Retry multiplier at each layer; no cumulative check
Prevention	**Budget gate** — per-conversation cap ($2) limits total to ~$76

▶ **Additional anecdotal reports (self-published sources)**

Category B: Action Failures

Agents that take wrong, excessive, or unauthorized actions — where the damage is in the consequence, not the tokens.

B1. 200 wrong emails — $1.40 in tokens, $50K+ in damage ⚙️

A support agent sent 200 collections emails instead of welcome emails ↗. A prompt regression changed the template selection. Total model spend: $1.40. Business impact: 34 support tickets, 12 social media complaints, $50K+ in lost pipeline.

	Detail
Model cost	$1.40
Business impact	$50,000+ in lost pipeline
Root cause	No action-level enforcement — dollar budget was nowhere near exhausted
Prevention	**Action gate** — RISK_POINTS ↗ cap on email tool (50 points/email × 4 max = 200 points) blocks email #5

B2. Replit AI deletes production database

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.

	Detail
Model cost	~$2.00
Business impact	Production data loss, fabricated records
Source	TechCrunch, October 2025
Root cause	No pre-execution check on database mutation tools
Prevention	**Action gate** — database DELETE scored as Tier 4 action (50+ risk points), blocked without explicit authorization

B3. OpenAI Operator unauthorized purchase — $31.43

OpenAI's Operator agent made an unauthorized $31.43 purchase from Instacart ↗, bypassing user confirmation safeguards. The incident is also catalogued in the AI Incident Database ↗.

	Detail
Model cost	~$0.80
Business impact	Unauthorized financial transaction
Source	Washington Post ↗, February 2025; AI Incident Database #1028 ↗
Root cause	No pre-execution authorization for payment actions
Prevention	**Action gate** — payment processing scored as Tier 4 (50+ risk points), requires explicit budget allocation

B4. Accidental production deploy ⚙️

A coding agent, while debugging CI, triggers a production deployment ↗ with an untested fix. Total model cost: $0.80. Business impact: production downtime.

	Detail
Model cost	$0.80
Business impact	Production downtime
Root cause	No action-level gate on deploy tools
Prevention	**Action gate** — deploy tools scored as Tier 4 (100 risk points), gated separately from the dollar budget

B5. Slack data leak ⚙️

A support agent posts diagnostic information containing internal system names and another customer's tenant ID ↗ to an external customer-facing Slack channel.

	Detail
Model cost	$0.30
Business impact	Data exposure ↗, security review, possible compliance notification
Root cause	No distinction between internal and external channel tools
Prevention	**Action gate** — external Slack posting scored as Tier 3 (20 risk points), limited per run

B6. Jira ticket storm ⚙️

A workflow agent parses a 50-line stack trace incorrectly ↗, creates 50 tickets from a single trace. Across 10 error reports, hundreds of duplicate tickets flood the on-call team in 8 minutes.

	Detail
Model cost	$3.50
Business impact	On-call team flooded, incident response disrupted
Root cause	No per-run cap on ticket creation actions
Prevention	**Action gate** — ticket creation scored as Tier 3 (20 risk points), capped at 10 per run

Category C: Security Incidents

Attacks exploiting the agent tool layer — tool poisoning, supply chain, privilege escalation, and infrastructure exposure.

C1. postmark-mcp — silent email exfiltration

The first confirmed malicious MCP server ↗ in the wild: `postmark-mcp` silently BCC'd every outgoing email ↗ to an attacker-controlled address. It ran for weeks before detection. No user interaction required.

	Detail
Model cost	N/A (infrastructure attack)
Business impact	All outgoing emails exfiltrated
Source	Snyk, 2026
Root cause	No tool-call authorization layer; agent trusts any installed MCP server
Prevention	**Action gate + audit trail** — tool allowlist restricts which tools can be called; every invocation logged with full scope

C2. ClawJacked — WebSocket agent hijacking

Researchers demonstrated that malicious websites can hijack locally-running AI agents via WebSocket ↗, executing arbitrary tool calls through the user's agent session.

	Detail
Model cost	N/A (attack vector)
Business impact	Arbitrary action execution under user's identity
Source	Security research, February 2026
Root cause	No authentication between agent host and tool server
Prevention	**Scope isolation** — per-session budget limits blast radius even if session is compromised

C3. ClawHub malicious skills — 341 credential-stealing tools

Researchers found 341 malicious ClawHub skills ↗ designed to steal credentials, exfiltrate data, or execute unauthorized actions. Separately, the ClawJacked disclosure ↗ identified 71 additional malicious skills using WebSocket hijacking techniques.

	Detail
Scale	341 malicious skills (Koi Security) + 71 (ClawJacked)
Source	The Hacker News ↗, February 2026
Root cause	No vetting, signing, or sandboxing of community tools
Prevention	**Action gate** — tool allowlist restricts agent to vetted tools only; unknown tools blocked before execution

C4. Exposed MCP servers — zero authentication

Trend Micro found 492 internet-exposed MCP servers ↗ with no client authentication or traffic encryption. Separately, Knostic reported 1,862 exposed MCP servers ↗, sampled 119, and found all 119 exposed internal tool listings without authentication.

	Detail
Scale	492 exposed (Trend Micro ↗) + 1,862 exposed (Knostic ↗)
Source	Trend Micro ↗, Knostic ↗, 2026
Root cause	MCP protocol has no built-in authentication
Prevention	**Scope isolation** — even unauthenticated access is bounded by per-tenant budget; blast radius contained

C5. Tool poisoning — 84% success rate

The MCP-ITP benchmark ↗ achieved up to 84.2% attack success rate (ASR) in benchmark settings under auto-approval. Attacks include rug pulls (tool changes behavior post-install), schema poisoning (hidden instructions in descriptions), and tool shadowing (malicious tool overrides legitimate one).

	Detail
Success rate	84.2% with auto-approval
Source	MCP-ITP framework (Ruiqi Li et al., 2026)
Root cause	Agent trusts tool descriptions and auto-approves calls
Prevention	**Action gate** — per-tool risk scoring, tool allowlists, pre-execution authorization

C6. 30+ CVEs in 60 days

Security researchers documented more than 30 CVEs ↗ against MCP implementations in the first 60 days of widespread adoption. The average security score across 17 popular MCP server audits was **34 out of 100**.

	Detail
Scale	30+ CVEs, average security score 34/100
Source	AI Security Hub ↗, 2026 (secondary summary)
Root cause	Rapid adoption without security review
Prevention	**Audit trail** — every tool invocation logged; anomalous patterns detectable

C7. GitHub Copilot RCE — CVE-2025-53773

A vulnerability in GitHub Copilot enabled prompt injection to execute arbitrary code ↗ on developer machines.

	Detail
Impact	Arbitrary code execution
Source	CVE-2025-53773
Root cause	No isolation between model reasoning and tool execution
Prevention	**Action gate** — code execution tools gated as Tier 4, require explicit budget allocation

C8. Rogue agent collaboration

Researchers demonstrated ↗ that compromised agents in multi-agent architectures can coordinate to escalate privileges and compromise downstream systems.

	Detail
Impact	Cascading privilege escalation
Source	The Register, March 2026
Root cause	No per-agent budget isolation in multi-agent systems
Prevention	**Scope isolation** — per-agent budget caps prevent any single agent from exceeding its allocation, even if compromised

Category D: Multi-Agent and Systemic Failures

Failures that emerge from agent interactions, coordination, and systemic properties.

D1. UC Berkeley MAST — 41–87% failure rates

UC Berkeley's MAST study ↗ analyzed 1,600+ execution traces across 7 multi-agent frameworks and found 14 distinct failure modes with 41–87% failure rates. Failure categories: system design issues (44.2%), inter-agent misalignment (32.3%), task verification failures (23.5%).

	Detail
Failure rate	41–87% across frameworks
Source	UC Berkeley MAST ↗, NeurIPS 2025 Spotlight
Root cause	No per-agent or per-delegation budget enforcement
Prevention	**Scope isolation + budget gate** — hierarchical budgets (tenant → workflow → agent) bound each agent's spend and actions independently

D2. Google DeepMind — 17x error amplification

Google DeepMind research ↗ found that multi-agent networks amplify errors by 17x. A 95% per-agent reliability rate yields only 36% overall reliability in a 20-step chain.

	Detail
Amplification	17x error multiplication
Source	Google Research, January 2026
Root cause	Errors propagate and compound across agent boundaries
Prevention	**Scope isolation** — per-agent budgets ensure one agent's failure doesn't exhaust another's resources

D3. Silent failures — 200 OK masking wrong results

An agent returns HTTP 200 for every call, but the underlying data is wrong ↗. In multi-step workflows, the error propagates through 10+ downstream steps before anyone notices — because every step "succeeded."

	Detail
Detection time	10+ steps after the error
Source	Multiple production reports
Root cause	No validation between agent steps; success is measured by status code, not result quality
Prevention	**Audit trail** — structured logging of every action enables post-hoc analysis; **budget gate** — per-step caps limit how far a corrupted result can propagate

Category E: Industry-Scale Evidence

Statistics from research firms and industry surveys that quantify the systemic problem. These are not agent-specific incidents — they are broader AI adoption data points that provide context for the agent failures above.

Finding	Source	Year	Notes
64% of $1B+ companies lost >$1M to AI failures	EY AI Survey ↗	2025	Covers AI broadly, not agent-specific
By some estimates, more than 80% of AI projects fail to reach production	RAND Corporation ↗	2024	RAND cites the estimate; the underlying rate is debated
55% of organizations had not yet implemented an AI governance framework; among those that had, 46% used either a dedicated framework or extended another governance framework	Gartner ↗	2024	The 46% and 55% are not clean complements — different base populations
Over 40% of agentic AI projects will be canceled by end of 2027	Gartner forecast	2025	Forecast, not measured
Over 80% of firms reported no impact on either employment or productivity over the last 3 years	NBER ↗	2026	Broad AI adoption survey, not agent-specific

Control mapping

Every incident maps to one or more runtime controls that would have prevented it:

Control	What it prevents	Incidents prevented
Budget gate (pre-execution cost cap)	Runaway spend, loops, retries, fan-out	A1–A4, D1
Action gate (RISK_POINTS ↗)	Wrong actions, excessive actions, unauthorized actions	B1–B6, C1, C3, C5, C7
Scope isolation (per-tenant, per-agent)	Cross-tenant blast radius, concurrent overruns, compromised agent containment	A3, C2, C4, C8, D1, D2
Audit trail (structured event log)	Undetected failures, compliance gaps, incident reconstruction	C1, C6, D3
Atomic reservation (concurrency-safe)	TOCTOU races, double-spend, concurrent burst	A3, A4

No single control prevents all incidents. The four controls are complementary — cost, action, scope, and audit each address a different failure dimension.

What this means

The incidents in this report share three properties:

1. **The agent had the capability to act.** Every framework gave the agent access to tools — email, deploy, delete, purchase, API calls. The capability was granted at configuration time and never re-evaluated at runtime.

2. **No control existed between intent and execution.** The model decided to act, and the action happened. No budget check, no risk scoring, no scope verification. The gap between "the agent wants to do X" and "X happens" was empty.

3. **Detection happened after the damage.** Dashboards showed the cost spike, logs recorded the wrong email, alerts fired after the deploy. Observation is not prevention. By the time anyone noticed, the consequence had already persisted — emails sent, data deleted, money spent, trust eroded.

Runtime authority ↗ — the pre-execution control layer ↗ that decides whether an agent's next action should proceed — addresses all three. It fills the gap between capability and execution with a decision point that checks budget, scores risk, verifies scope, and logs the result before anything happens.

The regulatory frameworks converge on the same conclusion. The EU AI Act's Article 14 ↗ (for high-risk systems) requires human oversight with a stop mechanism. NIST's AI RMF ↗ requires controls proportionate to risk. OWASP's Top 10 for Agentic Applications ↗ identifies tool misuse, excessive authority, and cascading failures as critical risks. The incidents in this report are what these frameworks exist to prevent.

Methodology

Sourcing. Incidents were collected from public disclosures (TechCrunch, The Register, Snyk), research papers (UC Berkeley MAST, Google DeepMind, MCP-ITP), security advisories (OWASP, CVE database), industry surveys (EY, RAND, Gartner, NBER), and community reports (Hacker News, Reddit, Medium). Pattern-based scenarios (marked ⚙️) are constructed from real failure modes observed across production deployments and documented in the Cycles incident library ↗.

Limitations. This report has survivorship bias — only incidents that were publicly disclosed or studied are included. The actual incidence rate is higher. Cost estimates for pattern-based scenarios use documented pricing models but may not match specific deployment configurations. The "prevention" column represents which control category addresses the root cause — not a guarantee that any specific implementation would have caught the exact scenario.

Updates. This report will be updated quarterly as new incidents are documented. If you have an incident to report, contact the Cycles team or open an issue on the docs repository ↗.

Further reading

- What Is Runtime Authority for AI Agents? ↗ — the foundational concept
- AI Agent Governance Framework ↗ — mapping regulations to runtime controls
- AI Agent Risk Assessment ↗ — tool-level risk scoring methodology
- 5 Failures Budget Controls Would Prevent ↗ — detailed cost incident analysis
- 5 Failures Only Action Controls Would Prevent ↗ — detailed action incident analysis
- Zero Trust for AI Agents ↗ — OWASP mapping and policy enforcement
- MCP Tool Poisoning ↗ — supply chain attack analysis
- Why Multi-Agent Systems Fail ↗ — UC Berkeley MAST cost model

Related how-to guides

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

MORE FROM THE BLOG

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)

AI Agent Runtime Enforcement: Security and Cost
March 25, 2026
(https://runcycles.io/blog/ai-agent-governance-runtime-enforcement-security-cost-compliance)

Zero Trust for AI Agents: Why Every Tool Call Needs a Policy Decision
March 24, 2026
(https://runcycles.io/blog/zero-trust-for-ai-agents-why-every-tool-call-needs-a-policy-decision)

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