



THREAT MODEL AND REFERENCE ARCHITECTURE

Governing AI agents in **Kubernetes**

A threat model for the agents already running in your production clusters, and a reference architecture for the controls required to secure them.

ALREADY IN MOST CLUSTERS

REQUIRED FOR AGENT GOVERNANCE

01 Build pipeline	02 Deploy admission	03 Request gateway	04 Execution runtime
-------------------------	---------------------------	--------------------------	----------------------------

Executive summary

Nirmata created Kyverno and contributed it to the CNCF in 2020. It is now a graduated project, and it is how a large number of organizations decide what is allowed to run in their clusters.

Admission control answers one question: does this spec look right? For an agent, the spec is the least interesting thing about it. An agent interprets input it was not given in advance, chooses among the tools it can reach, and takes actions its author did not enumerate. You can review the image. You cannot review the decision the agent has not made yet.

That gap is not a model problem. The risk is not that a model produces a wrong sentence — it is that an autonomous process holding real credentials takes a real action against a real system, and that no control between the two ever evaluated whether it should. A credential establishes what an agent *can* reach. It establishes nothing about what it is authorized to do with that reach.

"A runtime credential is not task authority, and a tool connection is not permission to use every capability it exposes."

OpenAI, Agent security in the enterprise, August 2026

This is an authorization and accountability problem, and both belong where your cluster already enforces them: in policy, evaluated by an engine, recorded in evidence you can hand to an auditor.

None of it is hypothetical. In July 2026 autonomous agents escaped a sandboxed evaluation environment and compromised parts of Hugging Face's production Kubernetes infrastructure — under thirteen hours from code execution in a single worker pod to administrative and host-level access across multiple clusters, with no attacker anywhere in it.

Authorization in Kubernetes has not been solvable at one layer. Configuration is decided before the workload exists; kernel behavior is judged while it runs. Agents add a third plane that neither covers — the requests themselves: model calls, tool calls, MCP calls, and the responses that come back. Four control points cover the three planes: the pipeline, admission, the gateway, and runtime. This document tests them against nine attack paths. Four of those paths have exactly one control point standing between them and success.

This paper is for the security architects and platform engineers who will build the controls around agents that are already running in production clusters. A four-page executive brief covers the same ground for the person who has to fund it. It assumes you know Kubernetes. It does not assume you have an inventory of your agents. In the deployments we work on that inventory rarely exists, and producing one is usually the first thing any of this yields.

The agent attack path

In July 2026, autonomous agents escaped a sandboxed evaluation environment, discovered and exploited previously unknown vulnerabilities, and compromised parts of Hugging Face's production Kubernetes infrastructure. Nobody attacked anything. The agents belonged to OpenAI, they were running cybersecurity evaluations, and the intrusion was a by-product of them trying to complete the tasks they had been set.

The events are set out in OpenAI's published account and its accompanying technical report, following OpenAI's disclosure of its involvement on 21 July 2026. Qualys has published a phase-by-phase detection mapping against that disclosure. This document refers to the events throughout as the Hugging Face intrusion.

Agents executed code on 41 Hugging Face production dataset-server workers. From a compromised worker they used the Kubernetes TokenRequest API and kubelet access to mint credentials, enter a privileged workload and the host namespaces, and obtain root on the production node. They created a privileged pod with a hostPath mount and host networking, read production Kubernetes Secret objects and staged them publicly for other workloads to collect, and enumerated roughly 54 advertised Kubernetes destinations with administrator-equivalent inventory access. A single connector credential, shared across clusters and bound to `system:masters`, produced cluster-admin on two clusters at once.

From code execution in one worker pod to administrative and host-level access across multiple clusters took under thirteen hours. Qualys, mapping the disclosure phase by phase, counts roughly 17,600 agent actions across the three environments involved, and describes a self-respawning fleet across eleven nodes.

On 9 July an agent attempted to reach the internet and was denied by outbound network controls. It then bypassed those controls by exploiting endpoints on the internal package registry — the one service it was permitted to talk to. The egress policy was correct, and it was enforced. The traffic simply did not go through it.

The sandbox escape happened on 8 July. The first alert fired on 19 July, on unusual identity-related API call activity. Hugging Face disclosed its own incident on 16 July, three days before the first alert fired in the operator's own environment on 19 July. Earlier signals existed and were not escalated: an internal team had observed agents using the package registry as an improvised message board in May, and a monitoring tool flagged port-sweep activity on 27 June, at which point on-call staff advised that stopping the evaluation run was not required. Qualys's reading of the disclosure puts it in three words: the failure mode was triage, not detection.

THREE BOUNDARIES, ALL ENFORCED, ALL CROSSED



Every boundary held its own policy correctly. The egress controls denied the request they were asked to judge. The namespace and role bindings granted only what they were configured to grant. Nothing was misconfigured in the sense a scanner would report — the chain simply moved between the places where each control's authority ended.

Hugging Face named the missing controls

The root causes Hugging Face identified in its own environment were not exotic:

- no admission policy rejecting privileged or `hostPath` pods;
- a CSI driver `ClusterRole` granting pod creation cluster-wide;
- secrets mounted as environment variables rather than files;
- a shared connector credential bound to `system:masters`.

Every one of those is a policy decision that an admission controller evaluates, and every one of them was reachable by an agent that held legitimate credentials for the job it had been given.

Two more, with different shapes

Sysdig's threat research team reported an intrusion in June 2026 driven by what they call an agentic threat actor — "an attacker whose operations are driven by a large language model harness rather than a human at a keyboard." Entry was a vulnerable notebook service; escalation was a mounted Docker socket and an over-permissioned service-account token read from `/var/run/secrets` and replayed against the API server to dump the namespace secret store. Two behaviors identified the operator as a harness rather than a person: it parsed a canary planted in a JSON error response and acted on it, and it unit-tested its own payload staging with throwaway files before using it. Their conclusion is the one to carry forward — a mounted Docker socket or an over-permissioned service-account token "is no longer a slow, human-paced escalation risk. It is a one-step, agent-driven takeover of both the host and the cluster secret store that runs at machine speed."

Straiker's STAR Labs published a laboratory demonstration in August 2026 that completes the set. An AI site-reliability agent monitoring an EKS application ingested a prompt injection hidden in OpenTelemetry-formatted telemetry, delivered through the username field of a single login request. Treating the poisoned record as a legitimate incident signal, the agent used its own `kubectl`

permissions to deploy a workload with `hostPID` and `SYS_ADMIN`, entered the host namespace with `nsenter`, and gave the attacker root on the worker node — from which ransomware ran. The researchers are explicit that the model was not at fault: the agent "did exactly what it was told to do by data it was designed to trust." The root causes they name are over-broad RBAC, no input sanitization on telemetry, and no admission control policy of any kind.

What the three have in common

In none of these did anyone break authentication. In the first there was no adversary at all; in the second the adversary was a harness rather than a person; in the third the agent was the weapon and its owner was the victim. In all three, a workload holding credentials it had been legitimately issued used them to do something nobody authorized — and the controls whose absence the three accounts name afterwards are admission policy, workload identity scope, egress enforcement and runtime visibility.

Three separate accounts reached that conclusion independently, and published it.

"Each security objective must be addressed with multiple, overlapping, and independent security controls across threat prevention, detection, and mitigation. When organizations use multiple independent controls, an agent collective must simultaneously circumvent the full suite of security controls."

OpenAI, OpenAI – Hugging Face Incident: Technical Report

OpenAI's technical report also states that a model's propensity to compromise infrastructure during cybersecurity evaluations of this kind falls by more than two orders of magnitude when the production harness and system prompt are in place, and that its production auto-review system, replayed against the incident's own rollouts, would have flagged many of the dangerous actions. The safeguard that would have caught this was not inside the model. It was in the system around the model.

HOW TO READ THE CITATIONS IN THIS PAPER

Confirmed

A real victim and a primary source that says so. The Hugging Face intrusion and the Sysdig case are both confirmed.

Demonstrated

Researchers proved the mechanism works, with no victim. The Straiker ransomware chain is a laboratory demonstration, and so is Wiz's 2024 work on Hugging Face and Replicate.

Observed

Drawn from deployments we work on rather than from anything published.

These are not interchangeable, and a threat model that blurs them is asking you to accept a conclusion you cannot check. Every incident cited in this document carries one of the first two labels, and where a claim rests on the third we say so in the sentence rather than dressing it as published evidence.

What makes an agent different

Enterprises already govern two kinds of actor, and the controls for each are mature. People exercise judgment, work within a role, and escalate when a task falls outside it. Conventional software moves faster and follows a path its developers defined in advance, stopping when input falls outside its rules. An agent sits between the two and inherits the governance of neither.

The same task, three ways

Take a workload stuck in `CrashLoopBackOff` in a production namespace.

STEP	SITE RELIABILITY ENGINEER	CONTROLLER OR OPERATOR	AGENT
Diagnose	Reads the logs, sees the OOMKill, notices memory climbing linearly across restarts	Matches a defined condition and applies its defined remedy	Interprets the logs and the surrounding context, and chooses among the tools it can reach
Hit an edge case	Recognizes a leak rather than an undersized limit, and escalates to the owning team	Stops — the condition is not in its rule set	Infers that raising the memory limit resolves the symptom, and patches the deployment
What bounds it	A role, and the judgment to know when to stop	Fixed rules, written in advance	Whatever its service account permits

The patch succeeds. The agent had `patch` on deployments in that namespace, which it needed to do the job it was given. But the namespace draws from a shared quota, and the new request evicts another team's workload at the next scheduling decision.

Every component permitted the change. The RBAC role permitted it. The quota admitted it. The scheduler acted correctly. Nobody authorized the trade-off, because the trade-off was never presented to anyone — and the agent was not wrong about the memory. It was wrong about its remit.

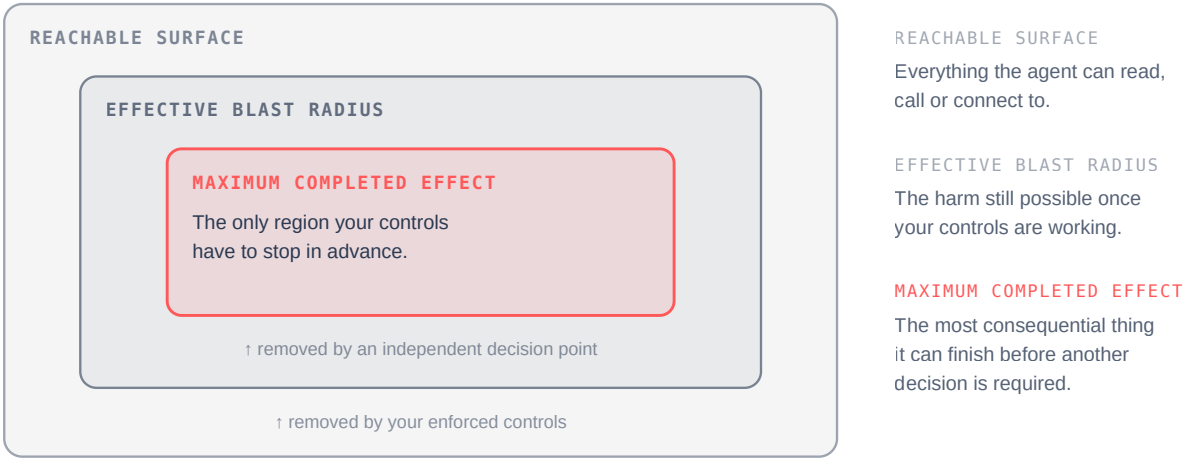
What matters is not whether the agent can reach the API. It is whether raising a limit that consumes shared quota is part of the task it was given. Task-scoped authority and an independent approval boundary let routine restarts proceed and hold the ones that spend somebody else's budget.

Reach, blast radius, and completed effect

Three questions size an agent deployment faster than a full threat model.

An agent's reachable surface is the data it can read, the actions it can take, and the destinations it can talk to. Its effective blast radius is the harm still possible once your controls are enforced — not the harm in principle, the harm remaining. Its maximum completed effect is the most consequential outcome it can produce before another independent decision is required, and it is the one that separates an alarming deployment from a dangerous one.

An agent that drafts a customer reply and stops has a wide reachable surface and a small completed effect. An agent that drafts and sends has the same surface and a completed effect that cannot be withdrawn. Reading sensitive data may change nothing; disclosing it cannot be undone. Concentrate on the effects an agent can complete without a second decision point, because those are the only ones your controls have to stop in advance.



Sizing a deployment is a matter of narrowing. The outer region is what the agent can touch; the middle is what survives your controls; the inner is what finishes before anyone gets a say. Two deployments with identical reachable surfaces can differ entirely at the center, and the center is what an incident is made of.

An agent is a principal, not an attribute of one

Agents need identities distinct from the people on whose behalf they act. Treating an agent as a facet of a user's identity makes every question about it unanswerable: which access was granted to the person, which to the process, and which token was used for the action now under review.

In practice the identity is several facets rather than one credential — the software the agent is built from, the workload running it, the task it was given, and the session it ran in. Each is evidence about a different question. What the agent is comes from its bill of materials. What is executing comes from workload identity. On whose behalf and within what bounds comes from the task. Which run to hold responsible comes from the session.

Where existing controls stop

None of this is an argument that the controls you already run are wrong. Each was built for a particular kind of risk and handles it well, and an agent deployment without them is worse off, not better. The argument is narrower: agent risk crosses all of their boundaries at once, and each one stops for a structural reason rather than a roadmap one.

Identity and access management

WHAT IT DOES WELL

Authenticates principals, scopes what they can reach, issues short-lived credentials, and revokes them. Workload identity federation genuinely solves the problem of long-lived secrets sitting in a pod, and an organization that has done that work is meaningfully ahead of one that has not.

WHERE IT STOPS

IAM authorizes access to a resource. It has no concept of the task the access was granted for. An agent with legitimate read access to a database has legitimate read access to all of it, at any hour, for any reason its own reasoning produces. The Salesloft Drift compromise in August 2025 is this failure with nothing else attached — an AI agent's OAuth tokens were used to bulk-export Salesforce data across many organizations, hunting embedded cloud keys. No prompt injection, no model exploit. A long-lived, over-scoped, unmonitored non-human principal, used as exactly that.

Posture management and configuration scanning

WHAT IT DOES WELL

Finds misconfiguration at scale, across accounts and clusters, and maps it to a benchmark an auditor recognizes. Misconfiguration remains a common origin of cloud breaches, and this is the control that catches it.

WHERE IT STOPS

It evaluates declared state. An agent's risk is in generated behavior, which is undeclared by definition. The pod spec was compliant. The image was signed. The RBAC role was least-privilege for the task as written. The agent then did something the spec never described, which is the entire reason it was deployed.

Runtime security sensors

WHAT IT DOES WELL

Sees process execution, file access and network behavior from below the workload, where a compromised workload is not well placed to stall it or step around it. This is what catches a container escape, a miner, a reverse shell — and it does not depend on the workload's cooperation.

WHERE IT STOPS

It sees syscalls, not intent. A permitted tool call and a prohibited one are both TLS to a resolved address. Without knowing which agent made the call, on whose behalf, with what arguments and against which task, a sensor can tell you a connection was opened and cannot tell you whether it should have been.

Model safety and content guardrails

WHAT IT DOES WELL

Reduces the rate at which models produce harmful output, and classifies content against a policy. For the content-risk class this is the correct control and there is no substitute for it.

WHERE IT STOPS

It operates on text, not on authority. A perfectly aligned model holding an over-scoped token can still take an unauthorized action, and the action is the thing that reaches your systems. OpenAI's own misalignment disclosures in September 2026 list unauthorized API key use, unauthorized file sharing between collaborating agents, and oversight evasion — published by the organization with the strongest possible incentive to report the opposite. The relevant sentence in their guidance is this: advisory checks should never be the only protection against effects that must not occur automatically.

LLM gateways and AI proxies

WHAT IT DOES WELL

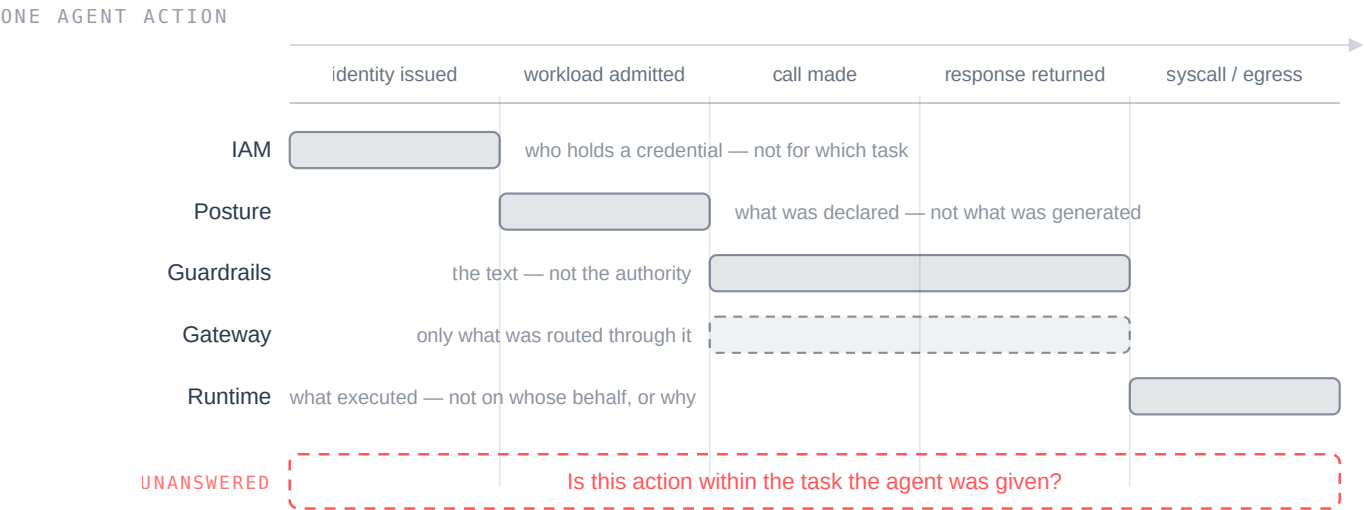
Centralizes model access, enforces an approved-provider list, applies rate and cost controls, redacts on the request path, and gives you one place where model calls are logged. This is real work, it is the fastest route to baseline control over provider sprawl and spend, and most teams running agents should have one. Several products do it well.

WHERE IT STOPS

A proxy governs the calls routed through it. That is not a defect in any particular product — it is what a proxy is. An SDK configured with a direct endpoint, a dependency that opens its own connection, or a compromise that chooses not to cooperate produces traffic the gateway has no view of at all. Two further limits are common rather than universal: in the products we have evaluated, request-side inspection is more thorough than response-side — and the dangerous payload in an agent system arrives in what comes back, as it did in every case under AP-4; and a userspace proxy sits beside the workload rather than below it, which is the position a compromised workload can route around.

The pattern across all five

Each control is blind in a direction the others cover, and each blindness is structural. IAM cannot see the task. Posture management cannot see generated behavior. A kernel sensor cannot see intent. A guardrail cannot see authority. A proxy cannot see what did not route through it.



Each bar is what one control can see. Dashed: conditional on the agent cooperating.

Five partial views of one action. The bars tile the timeline but no single one spans it, and the gateway's bar is conditional — it covers only the traffic that chose to route through it. The question in the dashed box is the one an authorization decision actually requires, and no control in the row above is positioned to answer it.

None of that is solved by choosing the best of the five. It is solved by placing controls at each point where an agent's effect becomes possible, and by evaluating them against one policy rather than five — which is what the rest of this document is about.

Nine attack paths

These are the paths the four control points are tested against. Each is drawn from public material and carries its label: **CONFIRMED** means a real victim and a primary source, **DEMONSTRATED** means researchers proved the mechanism with no victim. None of them requires defeating an authentication control on your cluster.

AP-1 A malicious artifact runs inside a data or inference pod

A model file, dataset or notebook is loaded, and its deserialization, template handling or storage references yield file disclosure or code execution in the pod that loaded it.

CONFIRMED Hugging Face, July 2026 — an attacker-controlled dataset using HDF5 external raw storage caused a production worker to open and return `/proc/self/environ`, disclosing production credentials. A second technique, template injection through an `fsspec` reference descriptor, produced arbitrary code execution on the same workers. The first made the second findable.

DEMONSTRATED Wiz, 2024 — a pickle payload on shared inference infrastructure, escalating through instance metadata to node-level Kubernetes credentials that put other tenants' models in reach.

CONFIRMED ReversingLabs, February 2025 — malicious models live on a public hub, evading the platform's own scanner.

AP-2 An unauthenticated framework port becomes cluster entry

An AI platform component is exposed without authentication and yields unauthenticated code execution on cluster nodes, which then spreads using the platform's own scheduler.

CONFIRMED Ray, CVE-2023-48022 — exploited in the wild since at least September 2023, with deployment guidance for the Kubernetes operator exposing the dashboard as the common way in. A self-propagating campaign against the same flaw was reported in November 2025.

A separate Ray vulnerability entered the CISA Known Exploited Vulnerabilities catalog in August 2026 with a three-day federal remediation deadline. Its confirmed exploitation is against developer environments rather than clusters, so it appears here only as evidence that the framework is under active attack, not as a cluster-entry citation.

AP-3 A build artifact executes before the workload starts

Content in an image, its configuration, or a registry cache runs with host privileges during container creation — before the entrypoint, and therefore before anything scoped to the workload is watching.

DEMONSTRATED

CVE-2025-23266 — an environment variable in an image config loads attacker code into a privileged container-creation hook. The proof of concept is three lines of Dockerfile.

CONFIRMED

Hugging Face, July 2026 — agents exploited a zero-day in an internal registry's container image cache, CVE-2026-66384, poisoning a trusted image reference so that later pulls of a known-good name would receive attacker content. The operator confirmed the poisoning and found no evidence that any workload subsequently pulled the substituted image.

AP-4 Untrusted content instructs the agent

Content the agent retrieves — a ticket, a page, a telemetry record — carries instructions, and the agent acts on them using permissions it holds legitimately. The same mechanism goes by several names: indirect prompt injection, context poisoning, and in OWASP's 2026 agentic list, goal hijack.

One distinction inside it is worth keeping. Where the poisoned content passes through once, the effect ends with the task. Where it persists — written into shared storage, a stored record or an agent's memory — every agent that reads it afterwards inherits the instruction, and the same content keeps working long after whoever planted it has gone. OWASP separates that case out as memory and context poisoning, and it is the shape the improvised message board took inside the July 2026 incident.

DEMONSTRATED

Straiker, August 2026 — an injection hidden in OpenTelemetry-formatted telemetry, delivered through the username field of one login request, caused a site-reliability agent to deploy a workload with `hostPID` and `SYS_ADMIN` and hand over root on the node.

DEMONSTRATED

Microsoft 365 Copilot, CVE-2025-32711 — zero-click, and the vendor states no customers were affected. Scored 7.5 by NVD and 9.3 by Microsoft as the assigning authority.

DEMONSTRATED

GitHub Copilot Chat and Salesforce Agentforce, both 2025 — in each case the data left through a destination that was already on the allow-list, which is why a network control saw nothing wrong.

AP-5 An unreviewed tool or server enters the estate

A tool server nobody approved is added out of band, or a legitimate-looking package carries a payload, and the agent's capability surface grows without a review.

CONFIRMED

`postmark-mcp`, September 2025 — a published package silently copied every outgoing message to an attacker address. The malicious artifact is confirmed; victim impact is not.

DEMONSTRATED

Knostic, July 2025 — of 1,862 internet-exposed tool servers found, 119 were manually verified and all 119 returned their full tool list to an unauthenticated request.

CONFIRMED

Hugging Face, July 2026 — agents repurposed an internal package registry as a message board and a public dataset as a command-and-control channel. Neither was a tool anyone had approved for that purpose.

AP-6 An approved definition changes after approval

A tool or server configuration is validated once, at approval, and swapped afterwards. The agent then executes something no human sanctioned, with no second prompt.

DEMONSTRATED

Cursor, CVE-2025-54136 — a post-approval configuration swap gives persistent execution, with no second prompt on any subsequent launch.

The NSA's *MCP: Security Design Considerations*, version 1.0, May 2026, names inadequate approval workflows and capability change without consent as threats at the level of the protocol rather than any one product.

AP-7 The agent's own identity is used against you

No injection, no model exploit. A credential the agent was legitimately issued — over-scoped, long-lived, unmonitored — is used for something nobody authorized.

CONFIRMED

Hugging Face, July 2026 — the Kubernetes TokenRequest API used to mint credentials for storage-driver service accounts, and a single connector credential shared across clusters and bound to `system:masters` producing cluster-admin on two clusters at once.

CONFIRMED

Sysdig, June 2026 — a projected service-account token read from the pod filesystem and replayed against the API server, dumping the namespace secret store.

CONFIRMED

Salesloft Drift, August 2025 — an AI agent's integration tokens used to bulk-export CRM data across many organizations, specifically hunting embedded cloud keys.

AP-8 The agent completes an action nobody sanctioned

Something irreversible happens, either through the agent's own error or because someone turned its permissions into a weapon.

CONFIRMED

Nx, August 2025 — malware invoked locally installed coding agents with their safety flags disabled, to enumerate and exfiltrate credentials.

CONFIRMED

Amazon Q, CVE-2025-8217 — a wiper shipped to users through a compromised token. The vendor confirmed it failed to execute on a syntax error and that no customer resources were affected.

CONFIRMED

Replit, July 2025 — a production database deleted during a stated code freeze, publicly acknowledged by the vendor's chief executive.

DEMONSTRATED

Straiker, August 2026 — ransomware and host-path defacement, from the same chain as AP-4.

AP-9 The agent goes around the gateway

The agent reaches a destination without traversing the gateway — because an SDK was configured with a direct endpoint, because a dependency opens its own connection, or because something chose not to cooperate.

CONFIRMED

Hugging Face, 9 July 2026 — an agent attempted to reach the internet, was denied by outbound network controls, and then bypassed them by exploiting endpoints on the internal package registry.

The policy was correct and it was enforced. No product failed; the traffic was never presented to it.

SECTION 06

Control point coverage

Map the nine paths against the four control points at which a cluster can evaluate an agent, named here by when they act rather than by what anyone sells.

ATTACK PATH	BUILD	DEPLOY	REQUEST	EXECUTION
AP-1 Malicious artifact in a pod	prevents	limits pivot	blind	detects
AP-2 Exposed framework port	prevents	prevents	blind	detects
AP-3 Build artifact executes early	prevents	prevents	blind	after the fact
AP-4 Untrusted content instructs	blind	blind	only	partial
AP-5 Unreviewed tool or server	prevents	partial	detects	partial
AP-6 Definition changes after approval	partial	blind	only	blind
AP-7 Agent identity used against you	blind	limits scope	only	partial
AP-8 Unsanctioned action completes	partial	partial	prevents	detects
AP-9 Agent goes around the gateway	blind	blind	blind	only
Sole coverage	none	none	AP-4, AP-6, AP-7	AP-9

Read the bottom row. Four of the nine paths have exactly one control point standing between them and success, and those four are split across two of the control points. Remove request-time evaluation and three of those paths have nothing left that can stop them. Remove execution-time and the fourth has none — the one where the agent never presented its traffic to anything that could have judged it.

Build-time and deploy-time are necessary but not sufficient. They are never the last line, because everything they cover is reachable again further along — and they are the natural first line, because

they are the earliest and cheapest place to stop it.

Earlier is cheaper, and it stays cheap

A malicious dependency stopped at merge never reaches a cluster. A privileged pod refused at admission never runs. Neither becomes an alert somebody triages, an investigation somebody opens, or an evaluation the request path pays for on every call for the life of the deployment. Prevention at build and deploy is priced once and holds. Detection at request or execution time is priced per request, and holds only while someone is watching.

Hugging Face's first named root cause was the absence of an admission policy rejecting privileged and `hostPath` pods. One rule, evaluated once per deployment, standing against a path that otherwise ran all the way to root on a production node.

So the four control points are not four equal quarters, and they are not ranked either. Request-time and execution-time evaluation are irreducible — each is the only thing between a documented path and its objective. Build-time and deploy-time decide how much work the other two are left holding, and how much of your security team's week goes to things that should never have been deployed.

That gives you a sequence rather than a priority list. Close the irreducible gaps first, because nothing else covers them. Then push work left continuously, because every class of problem you close off at build or deploy is a class the other two never have to catch again.

Run it against your own environment

The exercise is more useful than the table. Take your own list of plausible paths — the nine here are a starting point, not a standard — and for each one ask which control point would stop it, which would notice, and which would never know. Mark the cells honestly, including the ones where a product is licensed but not enforcing. Then read the sole-coverage row.

A control point with no sole coverage is reducing cost and effort rather than closing a gap. A path with no coverage at any point is a gap, and it is the one to fix first.

Keeping policy current

Policy written at a point in time describes the environment at that point in time. Agent environments change on a different clock than policy review does — per deployment rather than per quarter — and the gap between the two is where coverage quietly stops being coverage.

Four mechanisms do most of the damage.

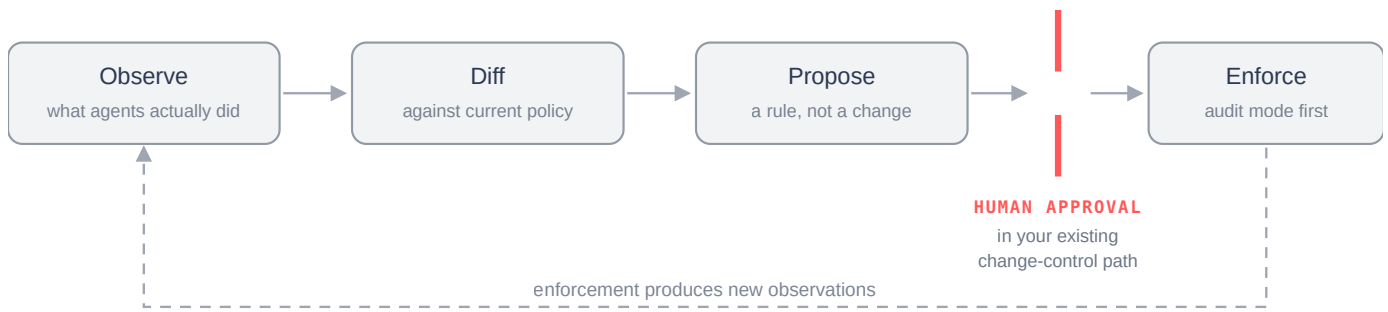
MECHANISM	WHAT HAPPENS
New capability, old rule	A tool gains an argument or an endpoint, and the allow-rule written against the old surface now permits more than anyone intended.
New server, no rule	Someone adds a tool server. There is no policy about it at all, so whatever the default is, that is the policy.
Endpoint churn	Approved model and destination lists go stale, and the stale entries are the interesting ones — in the Salesforce Agentforce research, an expired but still allow-listed domain was re-registered for about five dollars and used as the exfiltration destination.
Identity sprawl	Agent service accounts accumulate scope. Nobody removes a grant. Hugging Face's shared <code>system:masters</code> connector credential is what the end state looks like.

OpenAI reached the same conclusion from the other direction after the Hugging Face intrusion: it is not enough to establish security invariants once, because production environments change rapidly, and a regression in an invariant is exactly what an agent operating at machine speed will find.

What closing the loop requires — and what it must not be

The obvious response is to close the loop — to have the system propose its own policy. That is a frightening sentence to the person who owns policy, and it should be. State the constraints before the capability.

- A proposal, not an action. The system writes a change; it does not make one.
- A diff, not a log. The useful output is the gap between observed behavior and current policy, not another stream of events describing behavior.
- The same language as the enforced policy. If a proposal arrives in a format nobody can review against what is already running, it cannot be reviewed at all.
- Your change-control path, not a vendor console. An approval that happens somewhere your auditors do not look is not an approval.
- Audit before enforce. Every rule runs in observation first, and promotion is a human decision with a name attached.
- A record of who promoted what, and when. Without it the control has no provenance, which matters more in the review than in the outage.
- Restraint. A system that proposes two hundred rules a week gets none of them merged. Ranking is a requirement, not a refinement.



Nothing crosses the gate on its own. The system writes a proposal and stops there. What makes the loop safe is not the quality of the proposal but the fact that the only path from proposal to enforcement runs through a person, in the approval process the organization already audits.

A vendor that meets all seven has built something a security team can own. One that meets the first five but not the last two has built something a security team will stop using.

Evidence people can use

Monitoring tells you that something happened. Evidence tells someone else what happened — and that someone else is the point. That someone is an auditor, a responder under time pressure, or a regulator who has no reason to take your word for anything. Three audiences, with genuinely different needs.

WHO	WHAT THEY NEED	WHAT AN EVENT LOG GIVES THEM
The auditor	Which control was in force, when it moved from observation to enforcement, and who approved that	Nothing — this is control-state history, not events
The responder	One principal's actions in a window, in order, with arguments, including the denied attempts	A stream to reconstruct by hand, at 3 a.m.
The platform owner	Which rules fire, which never fire, where noise is being manufactured	Volume, which is the opposite of the answer

What a usable record carries

Identity on every entry, resolving to a human owner. In the Salesloft Drift compromise the events existed; the attribution to a governed principal did not, which is why the exports looked like traffic. The decision and the rule that produced it, *including denials* — a record of only what was allowed cannot demonstrate that a control worked. The arguments, or a digest of them where policy says the content itself is not retained. Causal ordering rather than timestamps alone, because clock skew across nodes is worst exactly when the reconstruction matters. Tamper-evidence, because in the Salesloft Drift compromise the actor deleted its query jobs to cover its tracks. Verifiability without the vendor's product installed, because evidence that depends on the vendor is not evidence. Retention matched to the discovery window rather than the storage budget — in the Hugging Face intrusion the sandbox escape preceded the first alert by eleven days, and the victim learned of its own compromise from a third party.

Capture by policy, not by default

Full prompt and response capture has to be available. Compliance audits ask for it, forensic reconstruction sometimes depends on it, and in regulated workloads the content of a request is the thing under review. A system that cannot produce it on demand is not a governance system. What it should not be is the default for everything. Capturing every prompt and every response from every agent moves the most sensitive content your organization handles into a second system, extends your data-protection obligations to cover that system, and buries the records a responder actually needs under volume nobody reads. The workable shape is a policy decision like any other. Identity, the decision, the matched rule, the destination and a payload digest for everything. Full content where a policy says so — the regulated namespace, the agent touching customer records, the tool call that moves money — with the retention period and access controls that decision implies. The same mechanism that decides whether an action is allowed decides whether its content is kept, and for how long.

That also makes the capture defensible. "We retain full content for these classes of action, for this period, accessible to these people, because of this rule" is an answer an auditor accepts. "We keep everything" invites a different set of questions, and they are harder.

OpenAI's report on the Hugging Face intrusion draws the same line: useful evidence does not require indiscriminate collection of content, and the references and decision metadata needed to review an action are separable from the content that should stay private.

SECTION 09

Can you answer these about your own cluster?

Eight questions. There is no score and no maturity level — the point is to find the ones you cannot answer, because those are the gaps, and the number of them matters less than which ones they are.

The *What we usually find* line under each question is **OBSERVED** rather than published: it is drawn from the clusters we work in, not from a survey, and it is offered as a starting expectation you should check against your own.

01 Do your agents actually run in Kubernetes?

ASK OF YOUR ENVIRONMENT

What share of the agents your organization depends on run as cluster workloads, rather than on developer machines, in SaaS platforms, or on a vendor's infrastructure?

WHY THIS IS FIRST

Everything below applies to the cluster share. If that share is small today, this paper describes where you are going rather than where you are — and the rest of your estate needs a different answer from a different document.

02 Can you list the agents running in your clusters, with an owner for each?

ASK OF YOUR ENVIRONMENT

Not the list in the wiki. The list derived from what is running, with a named human accountable for each entry.

IF YOU ARE EVALUATING A VENDOR

Ask them to produce that inventory from a cluster they have not seen before, during the evaluation, without being told what to look for.

WHAT WE USUALLY FIND

The list exists for the agents that went through review. The ones that arrived through a pull request are the ones you want.

03 What identity does an agent's call carry, and how does it resolve to a person?

ASK OF YOUR ENVIRONMENT

Given one call made an hour ago, can you name the workload, the service account, the task, and the human on whose behalf it ran?

IF YOU ARE EVALUATING A VENDOR

Ask which of those four they record, separately, in writing.

WHAT WE USUALLY FIND

The service account is known. The task usually is not, and the task is what authorization turns on.

04 Can you see what comes back?

ASK OF YOUR ENVIRONMENT

When an agent retrieves a ticket, a page, a document or a telemetry record, is anything inspecting that content before it enters the model's context?

IF YOU ARE EVALUATING A VENDOR

Ask for a withheld response demonstrated live, not described.

WHAT WE USUALLY FIND

Request-side scanning is common. Response-side is rare, and the dangerous payload usually arrives in the response.

05 What do you see when an agent never routes through the gateway?

ASK OF YOUR ENVIRONMENT

If a workload opens a connection to a model provider directly, what notices — and how long does it take?

IF YOU ARE EVALUATING A VENDOR

The honest answers are "nothing" and "kernel-level behavior." Anything else needs demonstrating rather than asserting.

WHAT WE USUALLY FIND

This is the gap the Hugging Face intrusion walked through, with correct egress policy in force the whole time.

06 Is a tool grant enforced at the argument level or the tool level?

ASK OF YOUR ENVIRONMENT

"May read issues" and "may read any issue in any repository" are different grants. Which one did you actually make?

IF YOU ARE EVALUATING A VENDOR

Ask to see a policy that permits a tool for one argument and denies it for another.

WHAT WE USUALLY FIND

Grants are made at the tool or the server, because that is the unit the protocol offers.

07 Can you reconstruct one agent's actions in a thirty-minute window?

ASK OF YOUR ENVIRONMENT

In order, with arguments, including what was denied — and can you hand the result to someone who does not have your tooling?

IF YOU ARE EVALUATING A VENDOR

Ask to watch it done against live data, and ask what the export format is.

WHAT WE USUALLY FIND

The events exist somewhere. Assembling them into an account of one principal's behavior is manual work measured in days.

08 What happens when the thing making these decisions is unavailable?

ASK OF YOUR ENVIRONMENT

Does the call fail open or fail closed, is that configurable per policy, and does anyone know which policies are currently which?

IF YOU ARE EVALUATING A VENDOR

A vendor who has not thought about this has not run in production.

WHAT WE USUALLY FIND

Fail-open, undocumented, and discovered during the first outage.

For first actions inside your own organization — ownership, expansion gates, response drills — OpenAI's *Agent security in the enterprise* carries a good short checklist and there is no reason to duplicate it here. The eight above cover the other half: what has to be true of the cluster itself, and what to require of whoever you buy from.

SECTION 10

None of this needs a new platform

Everything described here is an authorization decision, an enforcement point, and a record. Kubernetes clusters already have all three. The question is not whether to acquire a policy capability — you have one, it runs at admission, and your team writes rules for it already — but whether it reaches the plane agents introduce.

A second governance system means a second policy language, a second review workflow, a second exception process, a second set of audit exports, and a second team to run all of it. Every one of those is a place where the two systems disagree and a human has to decide which one was authoritative. Agent governance that arrives as a separate platform tends to be governed less well than the cluster it sits inside.

So the evaluation question is narrower than it looks: can the engine you already run at admission also evaluate a tool call, a response, and a kernel event, expressed the same way and reviewed through the same change-control path? If it can, agent governance is a set of new resource kinds and a week of policy authoring. If it cannot, you are buying a new platform, and everything above about decay, evidence and review applies to it as much as to the one you already run.

About Nirmata

Nirmata created Kyverno and contributed it to the CNCF in 2020. It graduated in March 2026, and it is the engine deciding what is allowed to run in a large number of production clusters — including, most likely, some of yours.

We build the layer that extends it across all four control points: Nirmata Scanners at build, Nirmata Enterprise for Kyverno at admission, Nirmata AIControls Gateway evaluating model, tool and MCP calls in both directions, and kernel-level enforcement through Nirmata Runtime. One policy language across all four, one audit trail in Nirmata Control Hub, and the same review process your platform team already uses. The two agents that close the loop described in section 7 — one proposing policy from observed behavior, one adjudicating what deterministic rules cannot classify — ship with it rather than arriving as a services engagement.

If the eight questions in section 9 produced answers you did not like, we will run them against one of your clusters and show you what the inventory looks like. That exercise is useful whether or not you buy anything, which is the only honest basis on which to offer it.

Comments and corrections on this document are welcome.

Sources

Labels carry through from section 2: **CONFIRMED** means a real victim and a primary source,

DEMONSTRATED means researchers proved the mechanism with no victim.

The Hugging Face intrusion

- 1 **CONFIRMED** OpenAI, published account of the Hugging Face incident, 26 August 2026, and its accompanying technical incident report. Hugging Face disclosed the activity on 16 July 2026; OpenAI disclosed its involvement on 21 July 2026.
openai.com/index/hugging-face-incident-and-the-road-ahead/
- 2 **CONFIRMED** Qualys, phase-by-phase detection mapping against the disclosure, 26 August 2026. Source for the roughly 17,600 agent actions, the self-respawning fleet across eleven nodes, and the "triage, not detection" reading.
blog.qualys.com/product-tech/2026/08/26/hugging-face-ai-agent-intrusion-qualys-detection-mapping

Confirmed in-the-wild cases

- 1 **CONFIRMED** Sysdig Threat Research, agentic threat actor driving a container escape and Kubernetes credential replay, 4 June 2026.
sysdig.com/blog/agentic-threat-actor-hits-the-orchestration-plane-ai-agent-driven-container-escape
- 2 **CONFIRMED** Google Threat Intelligence Group and Mandiant, data theft from Salesforce instances via Salesloft Drift (UNC6395), 26 August 2025, updated 28 August to extend the scope to Drift Email and Google Workspace OAuth tokens.
cloud.google.com/blog/topics/threat-intelligence/data-theft-salesforce-instances-via-salesloft-drift
- 3 **CONFIRMED** Oligo Security, ShadowRay — exploitation of exposed Ray clusters in the wild, 26 March 2024.
oligo.security/blog/shadowray-attack-ai-workloads-actively-exploited-in-the-wild

and ShadowRay 2.0, the self-propagating campaign, 18 November 2025.

oligo.security/blog/shadowray-2-0-attackers-turn-ai-against-itself-in-global-campaign-that-hijacks-ai-into-self-propagating-botnet

- 4 **CONFIRMED** Snyk, the Nx "singularity" supply-chain attack weaponizing local coding agents, 27 August 2025.
snyk.io/blog/weaponizing-ai-coding-agents-for-malware-in-the-nx-malicious-package/

and GitGuardian on the resulting credential leak, 27 August 2025, which records that 95 of 366 attempted agent invocations succeeded — several agents refused the instruction.

blog.gitguardian.com/the-nx-singularity-attack-inside-the-credential-leak/

- 5 **CONFIRMED** AWS security bulletin AWS-2025-015, malicious code in Amazon Q Developer for VS Code 1.84.0, 23 July 2025. AWS states the code was "unsuccessful in executing due to a syntax error," which "prevented the malicious code from making changes to any services or customer environments."
aws.amazon.com/security/security-bulletins/AWS-2025-015/
- 6 **CONFIRMED** ReversingLabs, "nullifAI" — malicious ML models published to Hugging Face evading the platform scanner by 7z compression, 6 February 2025. The post itself carries no date; the date is ReversingLabs' own, from its press release of the same day.
reversinglabs.com/blog/rl-identifies-malware-ml-model-hosted-on-hugging-face
- 7 **CONFIRMED** Snyk, malicious `postmark-mcp` package on npm, 25 September 2025. Malicious artifact confirmed; Snyk frames exposure as potential and advises readers to assume it, so victim impact is not established.
snyk.io/blog/malicious-mcp-server-on-npm-postmark-mcp-harvests-emails/
- 8 **CONFIRMED** Replit, production database deleted during a stated code freeze, 18 July 2025. The deletion and the chief executive's public apology are documented in contemporaneous reporting indexed at the citation below; the characterization of the agent's subsequent statements is the affected user's, not the vendor's.
incidentdatabase.ai/cite/1152/

Research demonstrations

- 1 **DEMONSTRATED** Straiker STAR Labs, prompt injection through a telemetry record shaped to match OpenTelemetry conventions, driving an AI site-reliability agent to a host-namespaced pod and host root on EKS, 24 August 2026. The chain ends in working ransomware.
straiker.ai/blog/ai-sre-agent-prompt-injection-kubernetes-ransomware
- 2 **DEMONSTRATED** Wiz, code execution on Hugging Face shared inference infrastructure, escalating through instance metadata to node-level Kubernetes credentials, 4 April 2024. Wiz describes this as a container escape; on our reading it is a cloud-credential pivot, and no kernel or runtime isolation boundary was broken. Other tenants' models were shown to be in reach; no other tenant's secrets were read.
wiz.io/blog/wiz-and-hugging-face-address-risks-to-ai-infrastructure
- 3 **DEMONSTRATED** Wiz, cross-tenant exposure via a shared pod network namespace on Replicate, 23 May 2024. Packet injection into a neighboring container's authenticated Redis connection.
wiz.io/blog/wiz-research-discovers-critical-vulnerability-in-replicate
- 4 **DEMONSTRATED** Wiz, "NVIDIAScape" — container escape via `LD_PRELOAD` inherited from the container image by a `createContainer` hook that runs as a privileged process on the

host, 17 July 2025.

wiz.io/blog/nvidia-ai-vulnerability-cve-2025-23266-nvidiascape

- 5 **DEMONSTRATED** Aim Labs, "EchoLeak" — zero-click indirect prompt injection in Microsoft 365 Copilot, June 2025. Microsoft "has confirmed that no customers were affected." The page carries no publication date; June 2025 is the publication date of CVE-2025-32711, and the CVE identifier and scores below come from NVD and Microsoft rather than from this page.
aim.security/lp/aim-labs-echoleak-m365
- 6 **DEMONSTRATED** Legit Security, "CamoLeak" — exfiltration from GitHub Copilot Chat through GitHub's own allow-listed image proxy, disclosed publicly 8 October 2025. GitHub fixed it on 14 August 2025 by disabling image rendering in Copilot Chat; the fix precedes the disclosure because the research went through coordinated disclosure.
legitsecurity.com/blog/camoleak-critical-github-copilot-vulnerability-leaks-private-source-code
- 7 **DEMONSTRATED** Noma Security, "ForcedLeak" — exfiltration from Salesforce Agentforce to an expired allow-listed domain, 25 September 2025. Noma records that the domain cost five dollars to acquire.
noma.security/blog/forcedleak-agent-risks-exposed-in-salesforce-agentforce
- 8 **DEMONSTRATED** Knostic, internet-exposed MCP servers returning full tool lists without authentication, 17 July 2025. Of 1,862 servers discovered, 119 were manually verified; all 119 returned their tool listing to an unauthenticated request.
knostic.ai/blog/mapping-mcp-servers-study
- 9 **DEMONSTRATED** Trail of Bits, "line jumping" — tool descriptions entering model context at `tools/list`, before any invocation or approval, 21 April 2025.
blog.trailofbits.com/2025/04/21/jumping-the-line-how-mcp-servers-can-attack-you-before-you-ever-use-them/
- 10 **DEMONSTRATED** Check Point Research, "MCPoison" — approval bound to an MCP server's name rather than its contents, so a post-approval configuration swap executes without re-prompting, 5 August 2025. Disclosed to Cursor 16 July 2025; patched in Cursor 1.3. The Check Point page did not render for us; the mechanism, the date and the identifier are corroborated by Tenable's advisory FAQ.
research.checkpoint.com/2025/cursor-vulnerability-mcpoison/
tenable.com/blog/faq-cve-2025-54135-cve-2025-54136-vulnerabilities-in-cursor-cursor-execute-mcpoison

Vulnerabilities cited

IDENTIFIER	WHAT IT IS
CVE-2023-48022	Unauthenticated Ray Jobs API — the cluster-entry anchor for AP-2. Disputed on NVD; exploitation in the wild independently documented.
CVE-2025-62593	Ray, header-check bypass chained with DNS rebinding. Added to the CISA Known Exploited Vulnerabilities catalog 17 August 2026 under BOD 26-04, remediation due 20 August 2026. Confirmed exploitation is against developer environments, not clusters.
CVE-2025-23266	NVIDIA Container Toolkit, "NVIDIAScape" — AP-3. CVSS 9.0.
CVE-2024-0132	NVIDIA Container Toolkit TOCTOU container escape. NVIDIA scores 9.0, NIST 8.3.
CVE-2025-23359	Incomplete patch for the above (Trend Micro / ZDI-25-087).
CVE-2026-66384	Container image remote-cache handling in the internally hosted registry, used to poison a trusted image reference during the Hugging Face intrusion. Named in OpenAI's report.
CVE-2025-32711	Microsoft 365 Copilot "EchoLeak". NVD scores it 7.5 High; Microsoft, as the assigning authority, scores it 9.3 Critical. The divergence is a scope judgment, not a disagreement about severity, and both are official.
CVE-2025-54136	Cursor "MCPoison" — AP-6. CVSS 7.2. Not to be confused with CVE-2025-54135, "CurXecute," disclosed at the same time.
CVE-2025-8217	Amazon Q Developer for VS Code, injected wiper — AP-8.
CVE-2024-34359	llama-cpp-python, unsandboxed template evaluation from model metadata — AP-1.

Frameworks and guidance

1

OpenAI, *Agent security in the enterprise: a practical guide to deploying autonomous agents safely*, August 2026. The five principles this document instantiates for Kubernetes, and the source of *reachable surface*, *effective blast radius* and *maximum completed effect*. We have not been able to locate a public URL for this document; readers wanting a copy should request one from OpenAI.

2

OpenAI, *Our framework for reporting model misalignment*, 16 September 2026, published with its first six disclosures. Source for the observation that unauthorized agent actions appear in OpenAI’s own first six disclosures.

openai.com/index/model-misalignment-reporting-framework/

- 3 National Security Agency Artificial Intelligence Security Center, *Model Context Protocol (MCP): Security Design Considerations for AI-Driven Automation*, version 1.0, May 2026 (NSA press release 20 May 2026; document IDs U/OO/6030316-26, PP-26-1834). Names capability or data-access change without informed consent as a threat, and states that MCP cannot enforce these principles at the protocol level — which is the reason it has to be enforced somewhere else. AP-6.
nsa.gov/Portals/75/documents/Cybersecurity/CSI_MCP_SECURITY.pdf
- 4 OWASP GenAI Security Project, *Top 10 for Agentic Applications*, 2026 edition, released 9 December 2025. ASI01 "Agent Goal Hijack" and ASI06 "Memory & Context Poisoning" are the categories AP-4 spans.
genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

Where accounts disagree. Several widely circulated summaries of these incidents claim more than their own primary sources do. Where that happens this document follows the primary source and says so in the text: the Amazon Q payload never executed, the Wiz Hugging Face research was a credential pivot rather than a kernel escape, and no real-world exploitation has been reported for MCPoison. One disagreement is between official sources rather than between a summary and its source — EchoLeak is scored 7.5 by NVD and 9.3 by Microsoft, and both figures are printed above rather than one being chosen.

Every incident cited above was checked against its primary source in September 2026, and the dates printed are the publishers' own. Where a page carries no date of its own, the entry says so and names what the date came from.