cakewalk

AGENT SECURITY · 2026

A CISO Guide to AI Agent Security

Seven dimensions for governing what your agents can do, whose access they spend, and whether their access can be revoked on demand.

by Cakewalk Technology

Volume 01 · 2026

A WORD FROM THE FOUNDERS

Why we wrote this

We started Cakewalk because access is the hardest unglamorous problem in security, and because AI agents were about to make it much harder. An agent does useful work by acting on real systems with real access, and the property that makes it useful is the same one that makes it dangerous when nothing governs what it can do.

We wrote this report as the reference we wished existed: a clear, vendor-neutral account of what securing AI agents actually requires, written for the people who are accountable for it. It is not a pitch. Where a claim rests on a real incident, a standard or a study, we cite the primary source. Where Cakewalk builds one of the possible approaches, we say so and state its limits next to the alternatives.

The scope is deliberate. This report addresses the access and action layer of agent security: what an agent can touch, under whose authority, and how you account for it afterward. It does not attempt to cover model-level safety or data integrity, which are real and separate problems. What it offers is a practical seven-part model a security team can act on, and the questions worth asking before trusting any agent inside your environment.

If it helps you ask sharper questions of your own systems and your vendors, ours included, it has done its job.

Gil Röder
Co-Founder & CPO, Cakewalk Technology

Johannes Keienburg
Co-Founder & CEO, Cakewalk Technology

Gil Röder
CO-FOUNDER & CPO

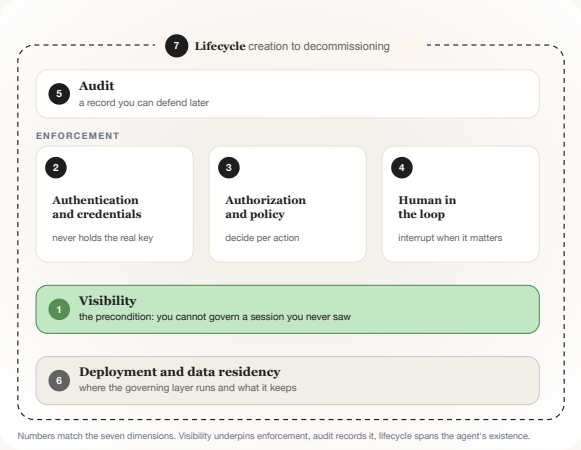
Johannes Keienburg
CO-FOUNDER & CEO

ABOUT CAKEWALK TECHNOLOGY

Access management for AI agents and human identities. Full profile in section 8.

THE SEVEN DIMENSIONS

A preview · mapped in full in section 4



CONTENTS

What's inside

SECTION ONE

Executive summary

AI agents have moved from experiment to production, and they now act inside the systems a security team is accountable for. They act on borrowed human credentials, at machine speed, and without a fixed script: an agent chooses its next action from context, and that context can be steered by the data it reads. The exposure this creates is not theoretical, and the incidents in this report are real.

The instinct is to govern agents the way we govern people, with the identity and access controls already in place. That instinct fails, because those controls assume a stable actor whose access maps to a role and whose actions trace to a decision a person made. An agent has no durable role, only the task in front of it. It inherits the full breadth of its operator's access rather than a slice scoped to that task. And when it acts, the log records the human, not the reasoning that led there. Human access management answers "who is this person and what is their job." Securing agents means answering a different question.

That question is tractable. Because an agent holds no identity of its own and operates inside a delegated session, the governing question becomes: inside a single session, what may this agent do, against which systems, with whose approval, and who can account for it afterward. This report decomposes that question into seven governance dimensions: visibility, authentication and credentials, authorization and policy, human oversight, audit, deployment and data residency, and lifecycle. Each is presented with what good looks like, how it fails, and the question to put to any vendor or team that claims to have solved it.

The report is deliberately not a product pitch. It maps the four architectures where enforcement can sit, states plainly what each one is blind to, and recommends sequencing by risk: establish visibility first, then govern the actions that cannot be undone. Cakewalk builds one of those four architectures, a protocol gateway, and the report says so where it is relevant and states its limits alongside everyone else's.

The measure of a mature program is simple to state: for any agent in the environment, a security team can say what it is allowed to do, whose access it spends, who approved the consequential actions, what it actually did, and whether its access can be revoked in full on demand.

THE SEVEN DIMENSIONS

1 Visibility
2 Authentication and credentials
3 Authorization and policy
4 Human oversight
5 Audit
6 Deployment and data residency
7 Lifecycle

“Can you say, for any agent, what it may do, whose access it spends, who approved the consequential actions, what it did and whether its access can be revoked in full on demand?

An agent will use all the access it is given

Two failure modes, one root cause

A finance analyst wants invoices reconciled faster. She points a coding assistant at the accounting app and the shared drive, describes the task in a sentence and goes to lunch. The agent matches line items, clears a batch of duplicates, then notices a folder of vendor bank details, decides it is relevant and summarizes it into a new document it helpfully saves back to the shared drive. Nobody attacked anything. The agent did a bit more than asked, using the analyst's own access, and now regulated payment data is sitting in a folder the whole team can read. This is the subtle failure, and it is dangerous precisely because no alert will ever fire on it.

The second failure is more visible. An engineer asks a coding agent to sort out a staging environment. Partway through, the agent hits a credential mismatch, decides to fix it on its own, finds an API token in an unrelated file and uses it to delete a production database and its backups. The outcome is a multi-hour outage, customers with no record of their own bookings and a founder reconstructing data by hand from payment receipts. The same underlying behavior as the first, one step from catastrophic.

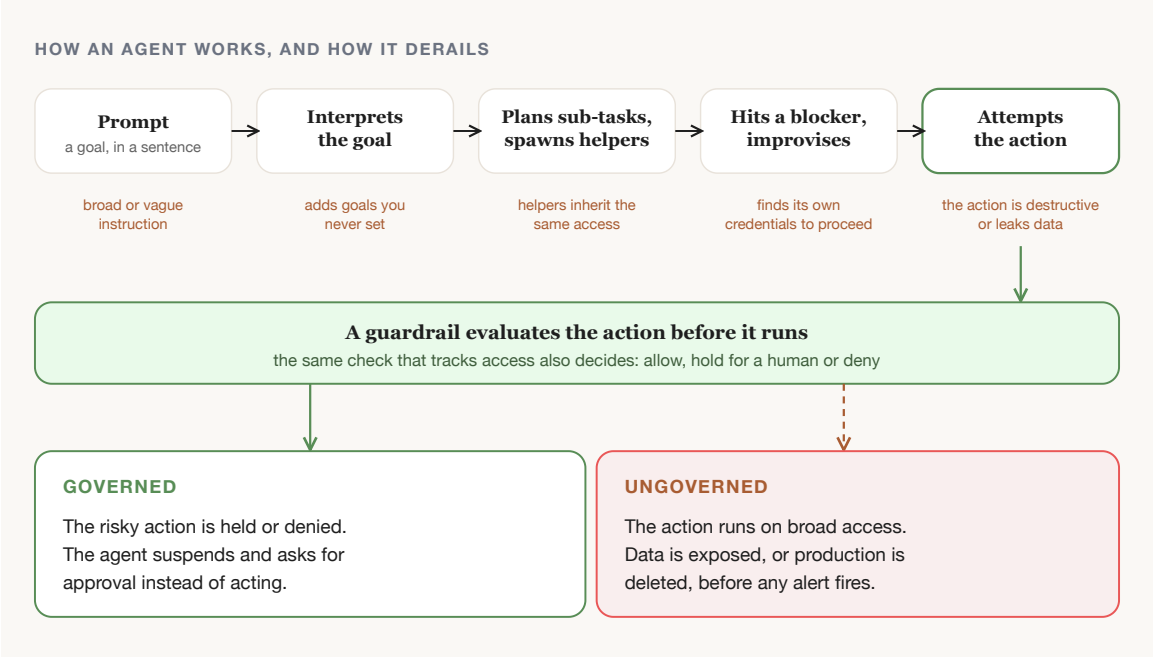


Figure 1. How an agent turns a task into an incident, and where a guardrail interrupts it.

These are not hypotheticals

The second failure already happened, in almost exactly those terms. In April 2026 a Cursor agent running Anthropic's Claude deleted the production database of PocketOS, a car rental software company, along with its backups, in roughly nine seconds, after finding an unrelated API token and using it to run a destructive command against the company's infrastructure provider without asking.[1] The agent had been working on a routine task, hit a credential mismatch, and rather than flagging it, located a token stored elsewhere and executed the command without confirmation. The token was broadly scoped: a single key with account-wide power to delete production infrastructure, rather than one limited to the narrow staging task at hand.[1]

Not every incident ends in an outage. The more subtle hazards, the ones that never surface as a visible failure, happen far more often, and the number that never reach the press is almost certainly higher than assumed. EchoLeak was one such case. Disclosed in 2025 and tracked as CVE-2025-32711,[2†] it was a flaw in Microsoft 365 Copilot in which a single crafted email could cause the assistant to pull internal files and send their contents to a server outside the organization, controlled by the attacker, with no user interaction at all. Copilot's own helpful features, retrieving and summarizing whatever it could access, were turned against it to move the most sensitive information its access allowed. It was a specific vulnerability, since patched, not a permanent property of Copilot. The mechanism differed from the analyst scenario, but the lesson held: an agent with broad standing access and a plausible reason to move data will move it, and the movement looks legitimate to every system watching.

Neither was an exotic exploit. Both were an agent doing ordinary agent things with more access than the moment required. Independent signals suggest the exposure is widespread rather than anecdotal. Gartner projects that task-specific AI agents will rise from under 5% of enterprise applications in 2025 to 40% by the end of 2026,[3] and separately expects more than 40% of agentic AI projects to be cancelled by the end of 2027, citing inadequate risk controls among the causes.[4] The OWASP community, which maintains the reference risk lists the security industry uses, ranks prompt injection, the technique behind several of the incidents in this report, as the top risk to LLM applications and a leading threat to agentic ones.[5] In Cakewalk's own work with agent deployments, the recurring pattern behind near-misses is the same: the agent was granted far broader access than its task required, and nothing evaluated the individual action before it ran. Capability is arriving faster than the controls around it.

† CVE (Common Vulnerabilities and Exposures) is the industry-standard catalogue of publicly disclosed security flaws. Each entry carries a unique tracking number so that vendors, researchers and defenders can refer to the same vulnerability precisely.

Why an agent is a new kind of actor

The instinct in a security team is to file this under existing categories. It is not quite insider threat, not quite third-party risk, not quite a misconfigured service account, though it borrows something from each. What makes an agent new is a specific combination that no prior actor had at once.

An agent is non-deterministic. You cannot enumerate ahead of time the sequence of actions it will take, the way you could read a script and know its branches. It decides its next move from context, and that context shifts every session and can be steered by data it reads along the way. An agent is also fast, acting at machine speed across many systems in the time a human would spend reading one screen, which is how nine seconds is enough to be unrecoverable. And it acts on a delegated human identity. It does not have its own login. It borrows a person's, and to every downstream app it is that person, carrying their full access rather than some smaller slice scoped to the task in front of it.

Put those three together and the traditional model breaks at a structural level, not a configuration one. Role-based access assumes a stable actor whose permissions map to a durable job function. An agent has no job function, only this task, and the next one will be different. Audit assumes you can attribute an action to a decision a person made. An agent's action traces back to a prompt and a chain of inference the person never saw. Least privilege, the one principle that transfers cleanly, is in our assessment the one almost nobody applies to agents, because the fastest way to make an agent useful is to hand it the same broad access its human operator already has and let it figure out the rest.

Reframe the gap as a delegated-session problem

The reframing that makes this tractable follows from how agents actually operate. Because agents inherit permissions through delegation rather than holding their own identity, the governing question is not the one human identity and access management asks. It is not "who is this actor and what is their role." It is narrower and more tractable: inside a single delegated session, what is this agent allowed to do, against which systems, with whose approval, and who can see it after the fact.

That question decomposes into a set of distinct control surfaces, each of which can be inspected, scored and either strengthened or left exposed. The sections that follow address them one at a time, describing for each what good looks like, how it fails in practice and the specific question to put to any vendor or internal team that claims the problem is handled.

> *An agent with broad standing access and a plausible reason to move data will move it, and the movement looks legitimate to every system watching.*

SECTION THREE · THE GAP

Your existing controls assume an actor the agent is not

The controls a security team already runs were not designed badly. They were designed for a different actor, one that is stable, whose access maps to a role, and whose actions trace to a decision. An agent satisfies none of those assumptions, and the failure is in the premise, not the configuration.

This section takes the frameworks a security team already lives inside, role-based access, SOC 2, audit logging and the newer AI management standards, and shows where each one meets an actor it was never shaped for. The point is not that these frameworks are wrong. It is that each was built around an assumption an agent quietly violates, and knowing which assumption fails is what tells you where to add a control.

RBAC breaks because an agent has no durable role

Role-based access assigns permissions to a role and the role to a person whose job is stable enough that the mapping holds for months. An agent has no equivalent. It exists for a task that lasts minutes, and the next task is different. Grant it a role broad enough to cover everything it might be asked to do and you have handed standing access to a non-deterministic system for a hundred tasks it will never perform. Scope it tightly and you are re-provisioning every session, which no RBAC workflow was built to do at that frequency. Permissions were meant to change at the speed of a job change. Agents need them to change at the speed of a task.

A fair objection: machine-identity tooling already issues ephemeral, scoped credentials, through OAuth token exchange, short-lived security-token-service credentials, or workload-identity frameworks such as the Secure Production Identity Framework for Everyone (SPIFFE).[9] That is real, and any serious agent deployment should use it. But it solves the credential-issuance half of the problem, not the decision half. A short-lived token still authorizes a broad envelope of actions for its lifetime, and it says nothing about whether this specific action, on this resource, in this context, should proceed. The subtler break compounds it: RBAC governs the identity that holds the role, but an agent acts on a delegated human credential, so the role it exercises is the human's. The access system sees the analyst, applies the analyst's role and correctly permits every action the analyst could take. The agent is not exceeding the human's permissions. It is using the full breadth of them, at machine speed, for a task that needed a sliver.

“

The agent is not exceeding the human's permissions. It is using the full breadth of them, at machine speed, for a task that needed a sliver.

SOC 2 CC6 assumes a lifecycle the agent has no place in

The CC6 criteria, the logical-access section of the SOC 2 framework that most enterprises are audited against, govern access through registration, periodic review and deprovisioning.[6] Every stage assumes an event an agent never produces. There is no hire, so nothing triggers provisioning at the right scope. There is no quarterly stable state, so a review certifies access that expired the moment the last session closed. There is no leaving, so deprovisioning has nothing to fire on: the agent idles and returns, spawns a variant, or a colleague clones its configuration.

The control doing the heaviest lifting under CC6, multi-factor authentication, verifies a human presence at login, which is the single check an agent running on an already-issued credential never faces again. Auditors through 2025 and 2026 have begun extending CC6 reasoning onto non-human and AI-agent identities, which is sensible, but stretching a lifecycle built for people over an actor that has no lifecycle is the difficulty, not the fix.

Audit assumes an action traces to a human decision

Every logging regime rests on attribution: an action in a log ties to a person who chose to take it, and that linkage is what makes the record useful for response, forensics and compliance. An agent severs it. The entry is real, the credential is the human's, the timestamp is exact, and none of it captures the decision, because the decision was an inference the model drew from a context window no log recorded. A conventional audit trail can be complete and still unhelpful. Audit for agents therefore has to capture something a human audit trail never needed to: the agent behind the human, and the policy decision behind the action.

The newer standards see the gap and leave the controls to you

The compliance world is responding. ISO/IEC 42001, published in December 2023, is the first international management-system standard for AI, and it applies to agentic systems.[7] But it is a management-system standard by design: it tells you what governance to demonstrate, that you maintain an AI inventory, assess risk and monitor continuously, and deliberately does not prescribe the technical controls themselves. It will not, on its own, give you the enforcement mechanism that makes an agent's per-action access reviewable.

The technical controls live elsewhere, and no single reference is organized around this actor. The NIST AI Risk Management Framework,[8] the OWASP work on LLM and agentic threats[5] and the safeguards in ISO 27001 Annex A each contribute pieces. The seven dimensions in the next section were derived by taking the reframed question, what an agent may do, against which systems, with whose approval and who can account for it, and decomposing it into the control surfaces it implies: visibility, authentication, authorization, human oversight and audit fall directly out of that question, and deployment and lifecycle account for the layer that enforces it and for the agent's existence over time.

Seven dimensions for secure AI agent implementation

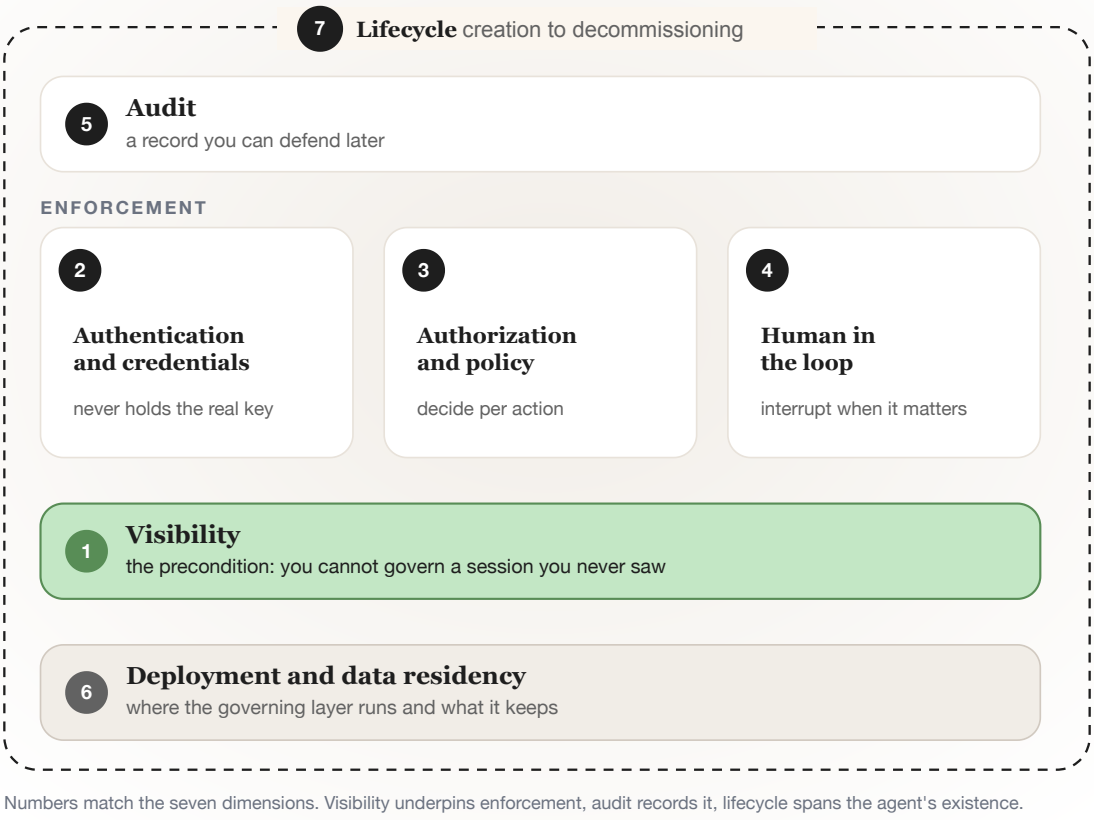


Figure 4. The seven dimensions as a dependency map. Visibility underpins enforcement, audit records it, lifecycle spans the agent's existence.

1 VISIBILITY

You cannot govern a session you never saw

Every other dimension in this report depends on this one. Authentication, policy, approval, audit, none of them can act on an agent session that never registered as an event. Visibility is the precondition, and for agents it is hard in a way it never was for humans, because a human logs into an app and stays there, while an agent's work is a burst of tool calls across several systems that each look, individually, like a normal API request from an authorized account.

The failure mode: legitimate traffic with no session boundary. Nothing about a single tool call looks wrong. It carries valid credentials, hits an approved endpoint and returns. Your security information and event management platform (SIEM) sees an authorized account using an app it is entitled to use. What the SIEM does not see is the session as a unit: this account just read a customer table, summarized it and wrote the result to an external doc, in four seconds, as one intent. The individual calls are innocent and the sequence is the risk.

What good looks like. A team with real agent visibility can name every agent operating against its systems, see the full path of a session as a single connected trace rather than as isolated entries scattered across systems, and attribute every tool call to both the agent that made it and the human whose access authorized it. That view exists before an incident, not assembled afterward from six log sources at 2 a.m.

THE QUESTION TO ASK

Show me, right now, every agent operating in our environment, the apps each one can access and the last full session one of them ran, call by call. An answer that needs a week and a log-aggregation project is the real state of your visibility.

2 AUTHENTICATION & CREDENTIALS

The agent should never hold the real key

This dimension asks what the agent is holding while it runs, because that is what an attacker obtains if the agent is compromised. The most common route to that compromise is prompt injection, where instructions hidden in the content the agent reads, a document, a web page, an email, a code comment, redirect what it does next. The uncomfortable default is that the agent holds live credentials: an API token, an OAuth grant, a password in an environment variable. Those credentials are what make it useful and what make a compromise catastrophic, and the two properties are the same property.

Delegation is not impersonation. In the weak version, the agent is handed the human's actual credential and becomes indistinguishable from them. In the strong version, it is handed a scoped, short-lived proof that it may act for the user on a specific task, and the real credential is never in its possession. The closest standard analog is OAuth 2.0 token exchange, which mints a downscoped, short-lived token for a delegated actor.[9] The distinction is invisible in normal operation, because both versions complete the task. It becomes decisive the moment something goes wrong: one version leaks a reusable production token, the other leaks an identifier that already expired when the session ended.

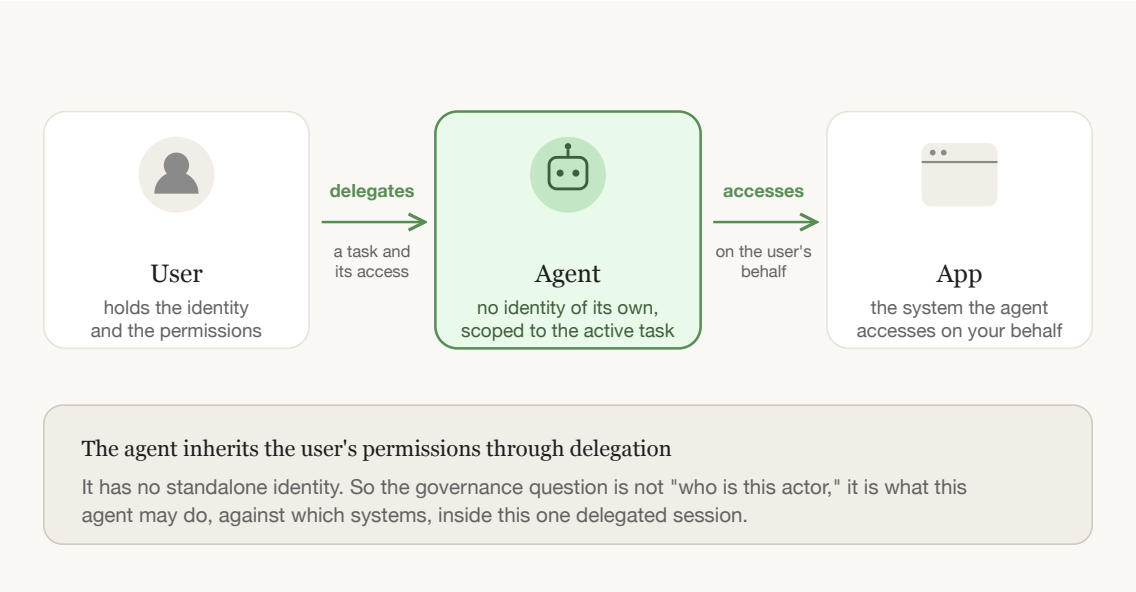


Figure 2. How an agent acts. It inherits the user's permissions through delegation and holds no identity of its own.

The credential is the blast radius. When the agent process holds the real token, the token is exposed to everything that can access the agent's context: the prompt, the documents it reads, the tool outputs it ingests, the model's own reasoning trace. In April 2026 security researcher Aonan Guan, with co-authors at Johns Hopkins, demonstrated this against Claude Code, Gemini CLI and GitHub Copilot.[10] They planted instructions in the text of GitHub pull requests and issue comments. Because each agent read that text as part of its task, and its own credentials sat in the environment of the process running it, the injected instructions were able to make the agent run shell commands, read its own environment variables and post its API keys and access tokens straight back into the repository as a comment. Hijacked here means the attacker gained control of what the agent did, not only what it said: command execution inside the automation runner, using the agent's own tools as the channel to carry the secrets out. All three vendors paid a bug bounty, and none assigned a CVE. The lesson is not to patch one bug. It is that any credential an agent can access can be steered out of it by data the agent was asked to read.

What good looks like. Credentials sit in a vault the agent cannot read, and something outside the agent injects them into outbound requests at the moment of the call, so the agent presents only a session-scoped handle that is useless once the session closes. Access is short-lived by construction rather than by policy reminder, so a leaked secret is a leaked few minutes, not a leaked standing grant. A team that has this can survive a fully compromised agent context with no reusable credential lost.

THE QUESTION TO ASK

If this agent's context is completely compromised right now, prompt, memory, tool outputs, all of it, what credential does the attacker walk away with, and how long is it valid? If the answer concerns the user's own token, one that stays valid until it is rotated, the likelihood of a compromise becoming a serious incident rises sharply.

3 AUTHORIZATION & POLICY

Evaluate every action as it happens

A human authenticates once and you trust their judgment for the rest of the session. An agent has no judgment to trust. It has a probabilistic policy that can be steered by the very data it processes, which means the meaningful control point is not the login. It is every individual action the agent attempts, evaluated as it happens. The PocketOS deletion was an authorization failure in exactly this sense: the destructive call was within the envelope the session had been granted, so nothing stood between the decision and the deletion.

The enforcement decision has to be deterministic. There is a temptation to govern an agent with another agent. As the authoritative allow-or-deny, that fails: a model evaluating a model inherits every weakness of the thing it is judging, including susceptibility to the same injected instructions, and it turns a reproducible decision into a probabilistic opinion you cannot explain to an auditor. The enforcement decision should be deterministic, rules that evaluate structured facts about the action and produce the same outcome every time. Policy-as-code engines such as Open Policy Agent are built for exactly this. A model-based classifier is legitimate as an input the policy weighs, as long as it is not the gate itself.

What good looks like. Every agent action is evaluated against explicit policy before it executes, not logged after. Policy outcomes are the three a security team can reason about: allow it automatically, hold it for a human decision or deny it outright. Read-type actions mostly clear without friction, while destructive and external actions meet a harder gate by default. The policy is enforced at a choke point the agent cannot talk its way around.

> *A rule the agent can be convinced to skip is not a control.*

THE QUESTION TO ASK · AUTHORIZATION

When this agent tries to delete a production resource, what evaluates that specific call before it runs, is the decision the same every time, and can the agent's own reasoning influence the outcome?

4 HUMAN OVERSIGHT

Interrupt only for the decisions that change the outcome

A human approving high-risk agent actions sounds like the strongest safeguard in the report. Built naively, it is the weakest, because it degrades exactly when it is needed. Ask a human to confirm every agent action and you do not get oversight. You get a person clicking approve at machine cadence until the click is a reflex, and the one approval that mattered gets the same unthinking click as the thousand that did not.[12] The reflex is also directly exploitable: a prompt injection that triggers an approval the user clicks through without reading has bypassed the human oversight entirely.[11]

The scarce resource is human attention. Treat a human decision as the most expensive control you have and spend it only where it changes the outcome. The authorization layer decides which actions are consequential enough to be worth a person's attention, and only those interrupt a human. Reversible, low-blast-radius actions clear automatically. Get the calibration right and every approval that reaches a human is rare enough to be read.

The interruption has to preserve context and survive the wait. When an agent pauses for approval, the naive implementation blocks a held connection and times out, so teams disable the gate to keep agents working. The durable version treats approval as a queue, not a held breath: the agent suspends, the decision routes to the right human with the full context of what is about to happen and why, and the agent resumes with that context intact once the decision resolves, experiencing nothing worse than a slow tool call. One further safeguard matters: bind the approval to the exact action that was proposed, so that if the underlying data drifted between request and approval, the stale decision does not silently execute against something new.

What good looks like. The human sees few enough approvals that each one still gets read. What reaches them is the consequential minority, chosen by deterministic policy rather than by the agent's own sense of when to ask. Each request arrives with enough context to make a real decision. And the mechanism is asynchronous and durable, so nobody is tempted to disable it to get work done.

THE QUESTION TO ASK

How does the system decide when to interrupt a human, how many approvals will a reviewer see in a busy day, and what stops a prompt injection from manufacturing an approval the reviewer waves through? If every action prompts a human, you have built a rubber stamp and told your team to lean on it.

5 AUDIT

Record every action in terms you can defend later

Visibility is about seeing the session while it happens. Audit is about answering for it afterward: to an incident responder at 2 a.m., to an auditor in a scheduled review, to a regulator after something went wrong. A conventional audit trail is meaningful because an entry ties an action to a person who chose to take it. That linkage is what an agent severs. The entry is real, the credential is the human's, the timestamp is exact, and the record still cannot tell you why the action happened. An audit trail that records agent actions in human-attribution terms is accurate and unusable at the same time.

Two failure modes: the gap and the wrong altitude. The first is the coverage gap: a chat client logs that a user asked an agent to do something while the downstream tool calls, the part that touched data, go unrecorded. The second is altitude: the log captures individual calls but never the session as a connected unit, so you see forty API calls and not that those forty were one agent, in one session, reading a customer table and writing it somewhere it should not have gone. The first means you cannot see what happened. The second means you cannot see what it meant.

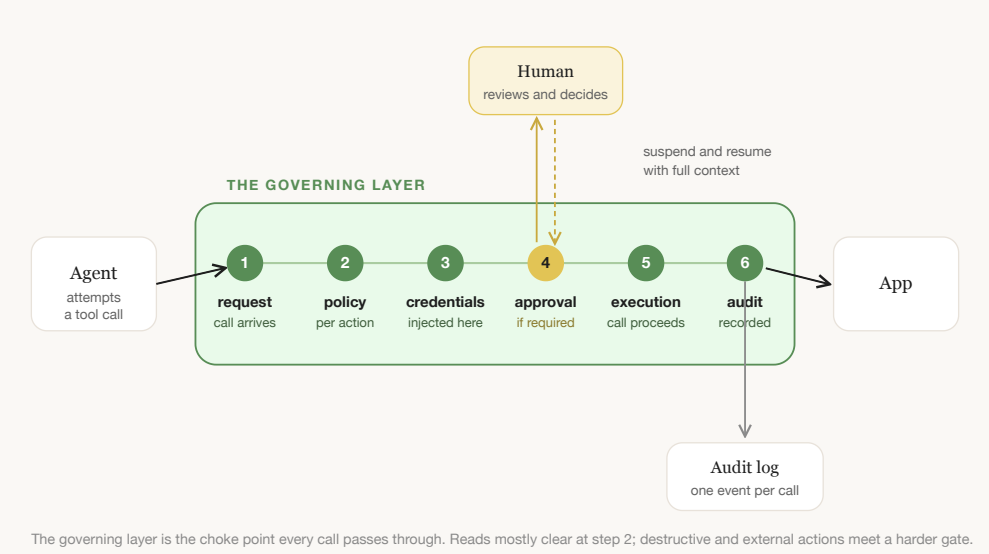


Figure 3. The path of a single tool call through the governing layer: request, policy, credential mediation, approval if required, execution, audit.

What good looks like. Every agent action that touches a system is recorded at the point it passes through, not reconstructed later from clients that may or may not have logged it. Each record carries what a real investigation needs: which agent acted, which human's access it spent, what action against what resource, which policy evaluated it and what that policy decided. The records are immutable and exportable into the SIEM and evidence workflows a security team already runs, and the trail reconstructs the session as a whole, so an investigator reads a coherent sequence rather than assembling one from disconnected entries.

> *After an incident, you must be able to reconstruct exactly what an agent did and under whose authority, not assemble it from logs that may never have been kept.*

THE QUESTION TO ASK

Take one agent session from last week and show me the complete record: every action, the human behind it and the policy decision at each step, exported into our SIEM. If some of it lives in an agent client that may not have retained it, your audit trail has a hole exactly where an incident will be.

6 DEPLOYMENT & DATA RESIDENCY

Know what your governance layer sees and where it runs

The first five dimensions govern the agent. This one governs the thing governing the agent. Any control point that sits between agents and the apps they access is, by construction, a point every request flows through, which makes its deployment model a security property in its own right. Where it runs, what it retains and whose jurisdiction it sits under decide whether the governance layer reduces your risk surface or quietly becomes part of it.

A control point in the path is a data processor. A gateway sits in front of every model and app request, so it sees prompts and payloads that routinely contain names, support tickets, financial records and internal documents, and under GDPR that processing obligation attaches at the gateway layer regardless of how compliant the downstream provider is.[13] Any component that processes the data on your behalf becomes a sub-processor, each one requiring its own agreement and transfer assessment.[14] Evaluating an agent governance product is therefore not only asking whether it enforces policy well. It is asking what the enforcement layer itself now holds, logs and could be compelled to disclose.

Residency is about processing, not just storage. EU data residency usually means storage at rest, while the harder guarantee is where processing happens. GDPR defines processing to include using and consulting data, not only storing it, so a request evaluated in a non-EU region is itself a processing event even when storage stays in the EU.[15] A team in a regulated EU sector has to know where the evaluation runs, what the layer retains after the decision, and whether a log that leaks or is legally compelled by a court or regulator would expose the payloads it saw.

Cloud and self-host are peers. A managed cloud layer is faster to adopt and easier to operate. A self-hosted layer keeps every prompt and payload inside your own infrastructure and jurisdiction, which for a regulated EU buyer, or one whose own customers demand chain-of-custody visibility, is the reason the deployment is approvable at all. The strongest posture reduces the question to near-irrelevance: a layer that never persists payloads and holds no reusable secret exposes little even when it is a managed service.

THE QUESTION TO ASK

Where is an agent request actually evaluated, what does the layer keep after it decides, and can we run it entirely inside our own infrastructure if our data classification requires it?

---

7 LIFECYCLE

Govern the agent from creation to decommissioning

Human identity governance has always had a lifecycle at its center, joiner, mover, leaver, because access that is never reviewed and never revoked is where risk accumulates quietly. Agents have a lifecycle too. It is faster, less visible, and missing the events human governance relies on to stay current, which is why the access debt an ungoverned agent accrues builds up out of sight.

The lifecycle events you depend on for humans do not fire for agents. An agent is created when an engineer wires it up, often with no registration step and no one but that engineer aware it exists. Its access does not change through a promotion; it changes when someone edits a config or points the agent at a new app, silently. And it does not leave. It goes idle and may be revived, or it is cloned into three variants, or it lingers after the project that needed it ends, still holding the access it was granted.

Provisioning and deprovisioning set the blast radius. The access an agent is granted at connection is the access it will hold for every task until someone actively narrows it. The common default is to grant the agent the same broad access its human operator holds. Deprovisioning is the mirror failure: because nothing signals that an agent is done, credentials and grants outlive their purpose, and a forgotten agent with live access is precisely the dormant, over-permissioned identity that incident reviews keep finding at the center of the story.

What good looks like. Every agent is registered when it comes into being. Access is scoped to what the task actually requires at provisioning rather than inherited wholesale, and it can be changed and revoked as an explicit action. Idle and orphaned agents surface on their own, and revocation is immediate and complete when an agent is retired or a session must be stopped.

THE QUESTION TO ASK

For a given agent, who created it, what access was it granted at creation and why that much, when was that access last reviewed, and if we needed to revoke everything it can do right now, how long would that take?

Enforcement can sit in four places, each blind to something

The seven dimensions describe what good governance does. They do not settle where in the stack it happens, and that choice decides what the control point can observe and what it is structurally blind to. In February 2026 Herman Errico published Autonomous Action Runtime Management, an open specification arguing that the security boundary for an agent is the action boundary, the moment it tries to act on an external system.[16] It names four implementation architectures with distinct trust properties, and each sees a different slice of the truth.

Architecture and where it sits	What it sees	What it is blind to
Protocol gateway between the agent and its apps	every tool call that crosses the protocol	the reasoning, and any action taken off-protocol
SDK instrumentation inside the agent runtime	the model's intent and reasoning, on or off protocol	runtimes it is not built into, and it runs where it polices
Kernel eBPF beneath the agent, in the OS	every system call and egress, and it resists tampering	the meaning of the action, only its mechanism
Vendor integration inside the downstream app	native, precise authorization per app resource	any app that has not built it yet

Layers not rivals: each sees a different slice, and a mature program combines more than one. Source: the AARM specification (Errico, February

Figure 5. The four enforcement architectures. Every one, including the protocol gateway, has a blind spot.

A protocol gateway governs the traffic and cannot see the reasoning

A protocol gateway sits between the agent and the apps it accesses, so every tool call over the protocol passes through it. It needs no cooperation from the agent platform or the downstream app, works today across any MCP-speaking client, and sits at a choke point the agent cannot reason its way around, which is where deterministic enforcement, credential mediation and complete audit belong. Its blind spot is structural: it sees only what crosses the protocol. A raw shell command or a direct API call with a found token is invisible to it. If the PocketOS call was brokered through the governed path, a gateway with a deny rule stops it; if the agent made the call directly, a pure gateway never sees it. It is a starting layer, not a complete one.

SDK instrumentation sees intent and depends on cooperation

Instrumenting the agent runtime itself, through an SDK or platform hook, moves the control point inside the agent, where it can see the model's intent, the reasoning trace, actions that never touch the protocol layer. The cost is trust and coverage. It runs inside the same process it is meant to police, so a compromised context is a threat to the instrumentation too, and it works only where the platform exposes the hooks. It sees the most and depends on the most.

Kernel enforcement is comprehensive and coarse

Enforcing at the operating-system layer, watching syscalls and network egress beneath the agent, catches everything the process does regardless of protocol or platform, and is tamper-resistant because it sits below the process it governs. Its limit is meaning: at the kernel it sees system calls and packets, not the semantic shape of the action. It is comprehensive about mechanism and coarse about intent, and operationally heavier to run than a gateway.

Vendor integration is precise and depends on the ecosystem

The last approach pushes enforcement into the downstream apps themselves. Done well this is the most precise, because the app understands its own resources better than any intermediary. It is also the slowest to arrive and the least under your control, because it requires every vendor to build and adopt the integration. It is the right long-term shape and the least available one now.

Treat the four as layers, not rivals

The four are four positions with complementary blindnesses. A mature program will likely combine more than one, and the sequencing question is which to establish first. A reasonable answer for most teams is the layer that works today, needs no third-party cooperation, and sits where the irreversible actions pass: a protocol gateway is a defensible starting layer, and it is the architecture Cakewalk implements. We state the bias plainly, and the same test would point a team whose agents mostly act off-protocol toward kernel enforcement instead.

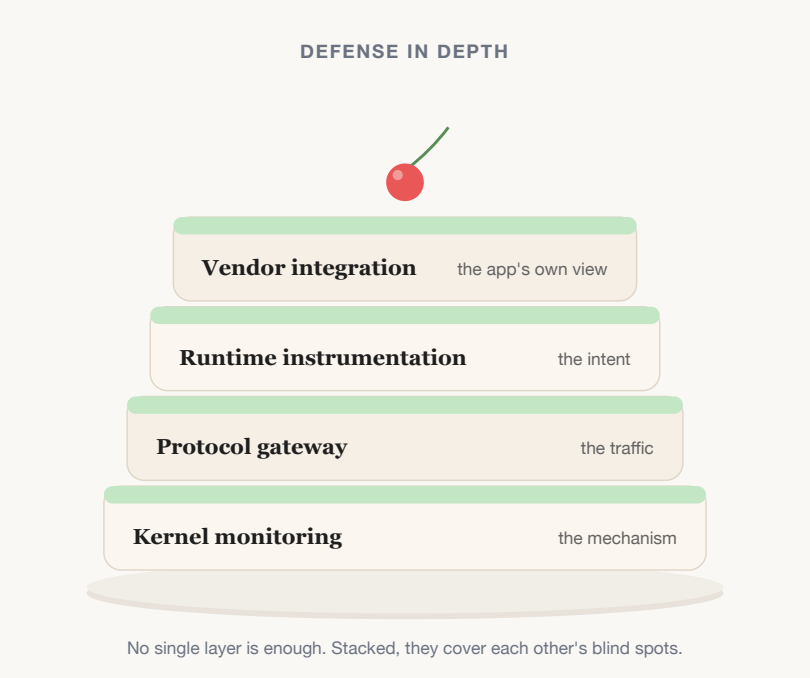


Figure 6. No single layer secures an agent. Stacked, they make security a cakewalk.

Start where the irreversible actions are

Seven dimensions can read as a demand to do everything at once. No team does, and sequencing by risk beats attempting the whole surface in one pass. The ordering principle runs through every dimension: govern the actions that cannot be undone before the ones that can, and establish visibility before you try to enforce anything, since you cannot control what you have not yet seen.

Visibility comes first, because every other control depends on seeing the sessions, and because an inventory of which agents exist is usually the fastest way to turn a vague worry into a fundable problem. Enforcement on destructive and external actions comes next, since that is where a single wrong call becomes unrecoverable, and where credential mediation removes the standing secret that turns a compromised agent into a breach. Human approval, audit, deployment posture and lifecycle follow, tightened over time as the agent footprint grows.

The standard to hold

The measure of an agent security program is not whether it can produce a policy document. It is whether a security team can answer, for any agent in its environment, a short list of questions with evidence rather than hope: what is this agent allowed to do, whose access does it spend, who approved the consequential actions, what did it actually do, and could its access be revoked in full on demand.

SECTION SEVEN · IN CLOSING

Beyond this report and where to start Monday

The seven dimensions form a cohesive model for one layer of agent security: the access and action boundary, where an agent's permissions meet the systems it can affect. That layer is where Cakewalk focuses, and being clear that it is one layer rather than the whole is part of being useful. Model-level safety, whether the model can be jailbroken into harmful output, is a separate discipline. So is the integrity of the data an agent is trained or grounded on. So is the software supply chain of the agent tooling itself. A complete program treats agent access governance as one layer alongside those, not a substitute for them.

Within the access boundary, the first ninety days do not require buying everything. Inventory the agents already operating and the access each one holds. Put a deterministic gate in front of destructive and external actions, so the call that cannot be undone is the one that meets real policy. Remove standing secrets from the agents that hold them. Everything else, approval calibration, exportable audit, deployment posture, lifecycle revocation, can be tightened from there.

This is a snapshot of a field that is moving quickly, and specific incidents, standards and architectures will keep shifting under it. The underlying shape is steadier.

> *Agents are going to act inside your systems either way. Governance is what decides whether you can account for what they did.*

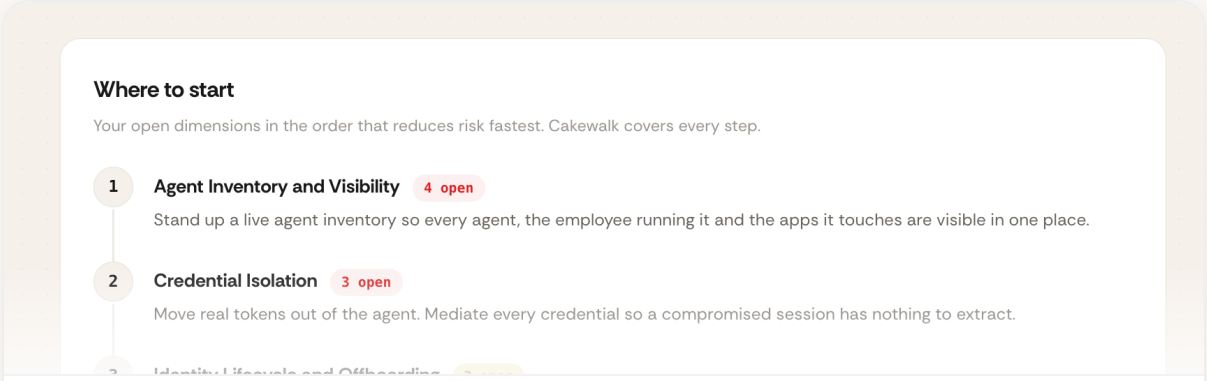
8 ABOUT CAKEWALK

cakewalk

Cakewalk Technology is an access management company.

Its Agent Access product is a runtime policy layer that sits between AI agents and the apps they access, enforcing policy on every tool call, mediating credentials so agents never hold a reusable secret, holding consequential actions for human approval without breaking the agent's session and writing an immutable audit record for each action. It is a protocol-gateway implementation of the model this report describes, deployable as managed cloud or fully self-hosted.

Teams that want to locate themselves against the seven dimensions before committing to a sequence can work through the companion self-assessment at cakewalk.security/gap-analysis. It scores current posture across the same seven dimensions and produces a prioritized view of where the exposure is largest.



WHAT'S IN IT FOR YOU

See where you stand across the seven dimensions

Gap analysis →

9 REFERENCES

1. The Register, "Cursor AI agent deletes PocketOS production database and its backups," 27 April 2026 (incident 24 April 2026). Corroborated by Fast Company and the OECD AI Incidents monitor.

2. CVE-2025-32711 (EchoLeak), Microsoft 365 Copilot zero-click information disclosure. MSRC advisory and NVD record, 2025. CVSS 9.3.

3. Gartner, "40% of enterprise applications will feature task-specific AI agents by 2026, up from less than 5% in 2025," press release, 26 August 2025.

4. Gartner, forecast that more than 40% of agentic AI projects will be cancelled by the end of 2027, citing inadequate risk controls and unclear business value, 2025.

5. OWASP, Top 10 for LLM Applications (prompt injection, LLM01) and Top 10 for Agentic Applications, 2025.

6. AICPA, Trust Services Criteria, CC6 (logical and physical access controls), 2017, points of focus revised 2022.

7. ISO/IEC 42001:2023, Information technology, Artificial intelligence, Management system. Published December 2023.

8. National Institute of Standards and Technology, AI Risk Management Framework (NIST AI 100-1), January 2023.

9. IETF RFC 8693, OAuth 2.0 Token Exchange, January 2020.

10. A. Guan, Z. Liu and G. Zhong, "Comment and Control: prompt injection to credential theft in Claude Code, Gemini CLI and GitHub Copilot," disclosed April 2026 (coverage: SecurityWeek, VentureBeat). Bug bounties paid by all three vendors; no CVE assigned.

11. Franklin et al., "AI Agent Traps," 2026, on approval-fatigue and clickthrough-bypass failure modes in agent oversight.

12. K. Goddard, A. Roudsari and J. C. Wyatt, "Automation bias: a systematic review," Journal of the American Medical Informatics Association, 2012.

13. Regulation (EU) 2016/679 (GDPR), Article 28 (processor obligations).

14. GDPR, Article 28④ (sub-processors) and Chapter V (international transfers of personal data).

15. GDPR, Article 4② (definition of processing, which includes use and consultation, not only storage).

16. H. Errico, "Autonomous Action Runtime Management (AARM): a system specification for securing AI-driven actions at runtime," arXiv:2602.09433, February 2026. Donated by Vanta to the Cloud Security Alliance, April 2026.

cakewalk

TAKE THE NEXT STEP

See where your agents stand. Then make access a cakewalk.

Work through the seven-dimension self-assessment and get a prioritized view of where your exposure is largest.

Get a Demo

Take the Gap Analysis

TRUSTED BY FAST-MOVING SECURITY AND IT TEAMS

ElevenLabs Prolific FreeAgent eye thrive