Policy and risk detection

Risk detectors, AI classifier, blocking, sensitivity.

- Overview
- MCP default action
- Approval for new MCP servers
- Risk classifier sensitivity
- Prompt capture
- Egress blocklist
- How a policy decision is made
- Secret scanning
- Credentials in context
- Personal data (PII)
- Lethal-trifecta enforcement
- The prompt-injection classifier
- Connection modes
- Device sessions
- Local MCP decision cache
- Blocked conversations
- Log retention
- Ungoverned-agent policy
- Credential-store reads
- Project configuration that runs code

Overview

Policy & risk detection

Every outbound AI request is scanned **before** it leaves, by layered detectors. Findings are recorded on the request's Activity row; depending on your settings they warn, report silently, or block.

Format detectors

High-precision pattern detectors: cloud keys (AWS, GitHub, Slack, Google, OpenAI, Anthropic), private keys, JWTs — plus a context detector for credential-looking values after words like "password:" that have no recognizable format.

Personal data (PII)

Emails, international phone numbers, IBANs (checksum-validated) and payment cards (Luhn-validated). Default action is **report** — PII in prompts is often legitimate; raise it to warn or block per your policy.

AI risk classifier

A local classifier plus an EU-hosted LLM tier evaluate prompt-injection and related risks with a per-tenant **sensitivity dial** (low / medium / high). A blocking verdict flags the conversation: the next request in that conversation is rejected pre-flight, and the block is visible (and clearable) in the console.

Dependency safety

Package references in prompts and AI responses are checked against registries and the OSV vulnerability database — nonexistent (slopsquat) packages and critical CVEs raise findings before `npm install` happens.

Actions per detector

Each detector has a per-tenant action: **allow** (off), **report** (silent record), **warn**, or **block**. Configure them on the Policy page; changes take effect immediately, no client changes.

MCP default action

This is the rule the Gateway applies to an MCP server that has **no explicit rule of its own**. Set it under **Policy ? MCP default action**.

Action	What developers experience	When to use it
Allow	Nothing changes	You want visibility first, controls later
Warn	A warning before the server is used	The usual starting point
Report	Nothing; usage is logged for review	You want a quiet audit trail
Block	The request is rejected	Locked-down environments

Per-server rules on **MCP Inventory** always override this default — so a strict default with a few allowed servers, or a permissive default with a few blocked ones, are both sensible shapes.

Where it applies

Enforced for Claude Code traffic today. Other tools are audited and appear in Activity; their MCP enforcement follows.

Recommended path

Start at **Warn**, watch **MCP Inventory** fill up with what your team actually uses for a week, set explicit rules for the servers you recognise, then decide whether to tighten the default. Starting at **Block** works but generates support requests on day one.

MCP default action

- ○ **Allow**
  MCP servers run without restriction.
- ◉ **Warn**
  Developers see a warning before using an MCP server.
- ○ **Report**
  Usage is logged for review; developers are not interrupted.
- ○ **Block**
  The Gateway rejects any request that declares an MCP tool.

Per-server rules on MCP Inventory override this default — allow specific servers through a strict default, or block individual ones under a permissive default. Enforced for Claude Code traffic today.

- ☐ **Require approval for new MCP servers**
  Servers nobody has reviewed yet are held at the pending action below.

“Reviewed” means any explicit per-server rule on MCP Inventory — setting one always takes over from this pending action.

AI risk classifier sensitivity

How aggressively the AI risk classifier acts on what it finds in outbound prompts. Medium (recommended) records medium- and high-severity findings as warnings; Low suppresses low-confidence signals for a quieter feed; High is the strictest. Findings appear on the Activity screen either way.

Policy ? MCP default action. Per-server rules on MCP Inventory override this default.

Approval for new MCP servers

Turn on **Require approval for new MCP servers** when you want unknown servers held back until someone has looked at them.

How it behaves

A server nobody has reviewed — meaning it has no explicit per-server rule on **MCP Inventory** — is enforced at the **pending action** you choose, instead of the MCP default action. Setting any explicit rule for that server counts as reviewing it, and takes over from the pending action immediately.

So the workflow is:

1. A developer starts using a new MCP server.
2. It appears in **MCP Inventory** with a *Pending approval* badge.
3. Until you set a rule, it's held at the pending action (typically `block` or `warn`).
4. You set `allow`, `warn`, `report` or `block` — and the badge disappears.

Why this exists

The MCP ecosystem is young and unvetted; a server added on Tuesday can do anything its tools describe. This setting turns "anyone can add anything" into "anything new waits for a human", without blocking the servers your team already relies on.

Risk classifier sensitivity

The risk classifier inspects outbound prompts for things that shouldn't leave — secrets, personal data, prompt-injection attempts, and exfiltration-shaped tool sequences.

Policy ? AI risk classifier sensitivity controls how aggressively it acts:

- **Low** — suppresses low-confidence signals, for a quieter feed.
- **Medium (recommended)** — records medium- and high-severity findings as warnings.
- **High** — the strictest setting.

Findings appear on the **Activity** screen either way; sensitivity changes what is acted on, not what is recorded.

Secrets are blocked regardless

A detected credential — an AWS key, a private key, a provider token — is blocked in the request path independently of this setting. Sensitivity governs the softer signals.

Reading the result

On **Activity**, a request with findings shows chips in the **Signals** column (`aws_access_key_id`, `email_address ×3`, and so on) and its outcome is `Blocked` if policy stopped it. Click through to the conversation when prompt capture is on.

Prompt capture

Off by default, and the setting that deserves the most thought before you turn it on.

What it does

Stores the actual content of conversations — what developers typed and what the model replied — so you can read them later on the **Conversations** screen.

Without it you still get everything else: who, when, which tool, which model, what the policy decided, what the detectors found. What you do not get is the text.

Before you turn it on

Tell your team. Not because the law necessarily requires a specific notice in your jurisdiction, but because discovering afterwards that their prompts were being stored is the fastest way to lose your developers' cooperation — and a governance programme that your engineers are working around is worse than none.

What protects the content

- **Redaction happens before storage.** Secrets found by the detectors are replaced with a marker naming the kind, so the stored transcript contains `[REDACTED:aws_access_key_id]` rather than the key. In the console these render as amber shield badges.
- **It follows your retention setting** and is purged with everything else.
- **Sentilai support cannot read it.** Support can see that a request happened and what the policy decided. The content is yours.

Turning it on

Policy ? Prompt capture. Conversations then start appearing on the Conversations screen. Until then that screen shows an explicit "Prompt capture is off" state with a link back to Policy, rather than an empty list that looks like a bug.

Egress blocklist

Policy ? Egress blocklist controls which network destinations the Endpoint Suite allows AI tools to reach directly, outside the Gateway.

Use **Download blocklist** to get the current list, and **Copy** to take it into your own tooling. **Send test** verifies the alerting path end to end.

This is a containment control, not a firewall: it constrains the tools Sentilai manages on machines running the Endpoint Suite. It is not a substitute for network egress controls at your perimeter.



The Egress blocklist section on Policy — generated from your providers and routed endpoints, with Download and Copy for your network team.

How a policy decision is made

Several things can have an opinion about a single request. This is the order they are consulted, so you can predict what will happen before you change anything.

For MCP tool calls

1. A **per-tool rule** — the most specific thing you can write.
2. A **per-server rule**.
3. The **pending action**, if approval is required for new servers and this one has no explicit rule.
4. The **tenant default**, `MCP default action`.

The first one that matches wins.

When one request names several servers

A single AI request can declare tools from more than one MCP server. The Gateway cannot partially block one API call, so **the most restrictive decision across all named servers applies to the whole request.** One blocked server blocks the request.

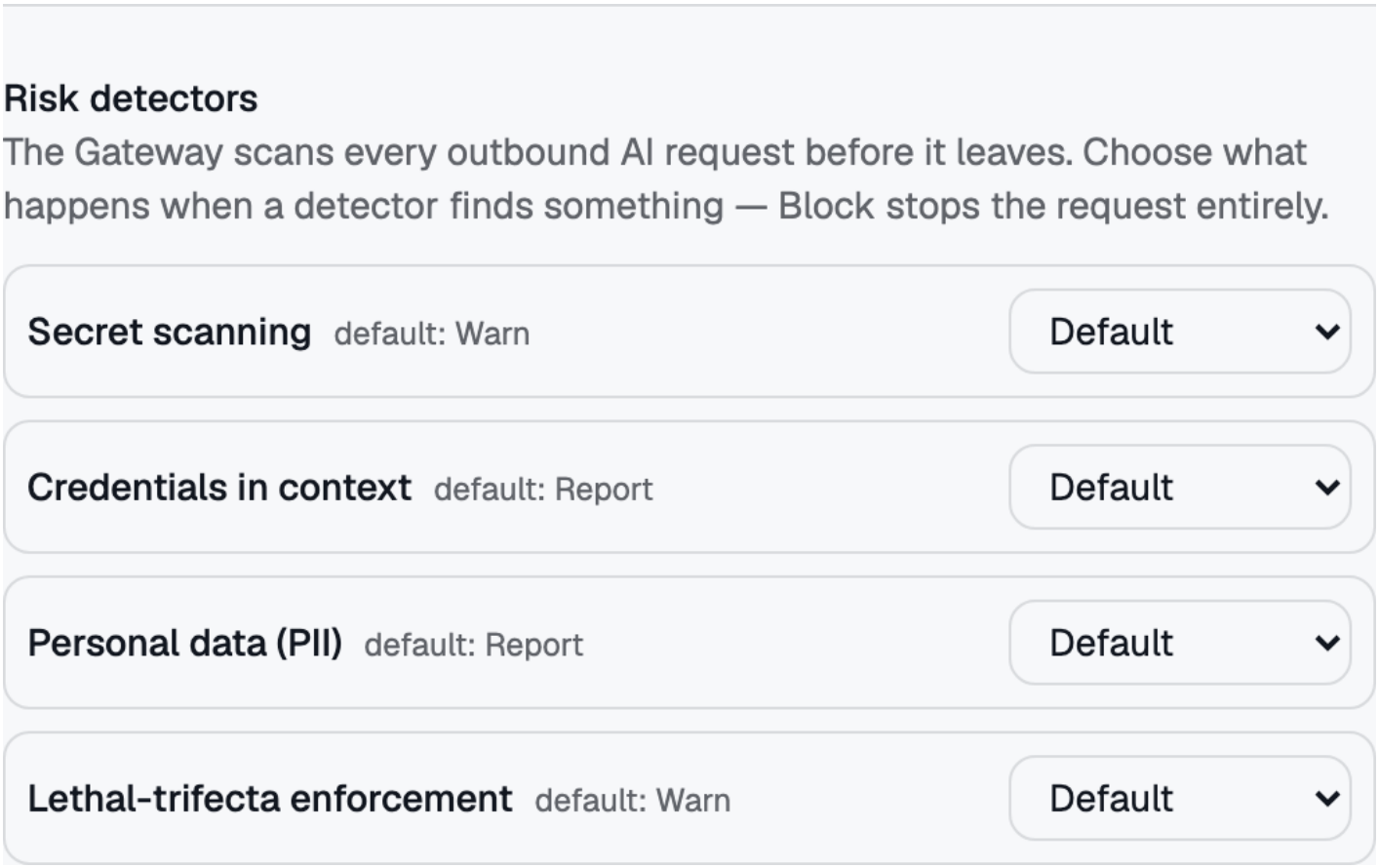
Risk detectors are separate

Detectors run on the content of the request, independently of MCP rules. Each has its own action, and the strongest action taken by any detector determines the outcome. A request can be allowed by MCP policy and blocked by secret scanning.

The actions

- **Allow** — nothing recorded beyond the normal audit row.
- **Report** — recorded as a finding, visible in Activity and Compliance. Nothing is interrupted.
- **Warn** — recorded more prominently and, where the path allows it, surfaced to the developer.
- **Block** — the request or tool call does not proceed.

Report is the setting you want while you are still learning what your team does. It gives you the same visibility as blocking with none of the disruption.



Risk detectors and their actions. The strongest action any detector takes decides the request.

Secret scanning

Stops credentials from leaving your organization inside a prompt. **Default: warn.** This is the detector most organizations move to **block** first, and the one where doing so is least controversial.

What it looks for

Recognizable, high-confidence credential formats:

- AWS access key IDs
- GitHub tokens and personal access tokens
- PEM private keys
- Anthropic API keys (`sk-ant-…`)
- OpenAI API keys (`sk-…`, `sk-proj-…`)
- Slack tokens
- Google API keys
- JSON Web Tokens

These are matched on shape, so the false-positive rate is low — but an example key in documentation your developer pasted will trip it, which is the right outcome anyway.

What is recorded

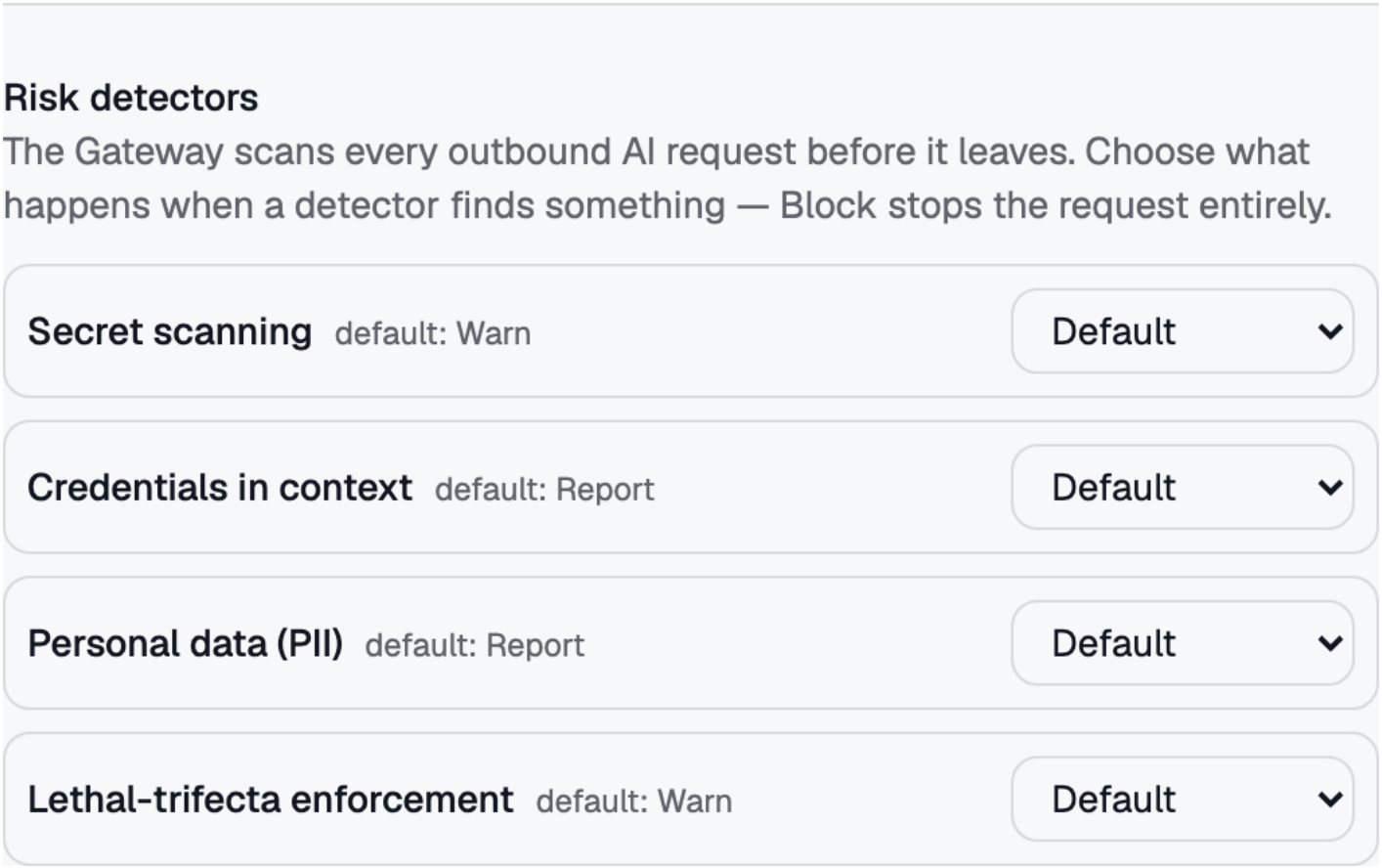
The kind and the count. Never the value. A finding says "one AWS access key id was present"; it does not store the key, an offset, or a preview. Building a security product that quietly collects a database of everybody's leaked credentials would be indefensible, so the detector is built so that it cannot.

The sensitivity dial does not apply

Detected secrets are handled according to **this detector's** action regardless of where the risk classifier sensitivity is set. The dial governs the semantic classifier, not pattern matching.

Setting it

Policy ? Risk detectors ? Secret scanning.



Secret scanning is the first row of Risk detectors. Each detector chooses its own action: Allow, Warn, Report or Block.

Credentials in context

Catches credentials that do not match a known vendor format — internal tokens, database passwords, connection strings. **Default: report.**

How it works

Rather than matching a shape, it looks for a credential-ish keyword close to a value that behaves like a secret. "password", "api_key", "token", "secret" next to something that looks like a value rather than a description.

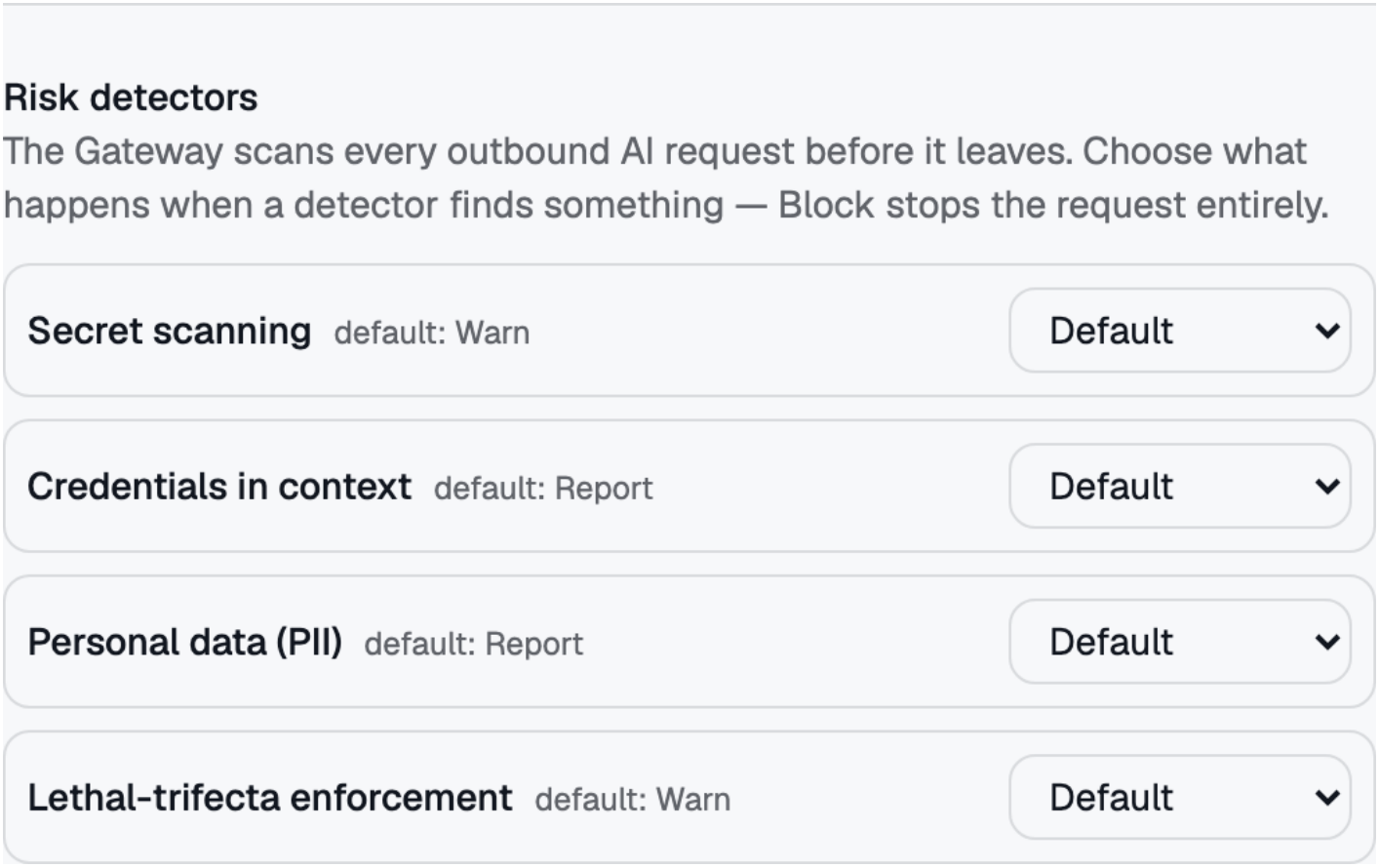
Why it defaults to report, not warn

This is a heuristic and it is wrong more often than the format matchers. It deliberately filters out the common innocent cases — a type annotation like `password: string`, a placeholder with no digits, a variable name with no value attached — but it will still sometimes flag a config example.

Run it in **report** for a while and look at what it actually catches in Activity before you promote it. If your team writes a lot of infrastructure code, expect noise.

What is recorded

Kind and count only, like every other detector. Never the matched value.



Credentials in context has its own row under Risk detectors, separate from secret scanning — it catches credentials arriving via attached files and context, not just the typed prompt.

Personal data (PII)

Flags personal data appearing in prompts. **Default: report.**

What it detects

- Email addresses
- International phone numbers, where a country prefix is present
- IBANs, **validated by checksum** rather than merely shaped like one
- Payment card numbers, **validated with the Luhn check**

The checksum validation on the last two matters: it means a random 16-digit number in a test fixture does not become a card-number incident.

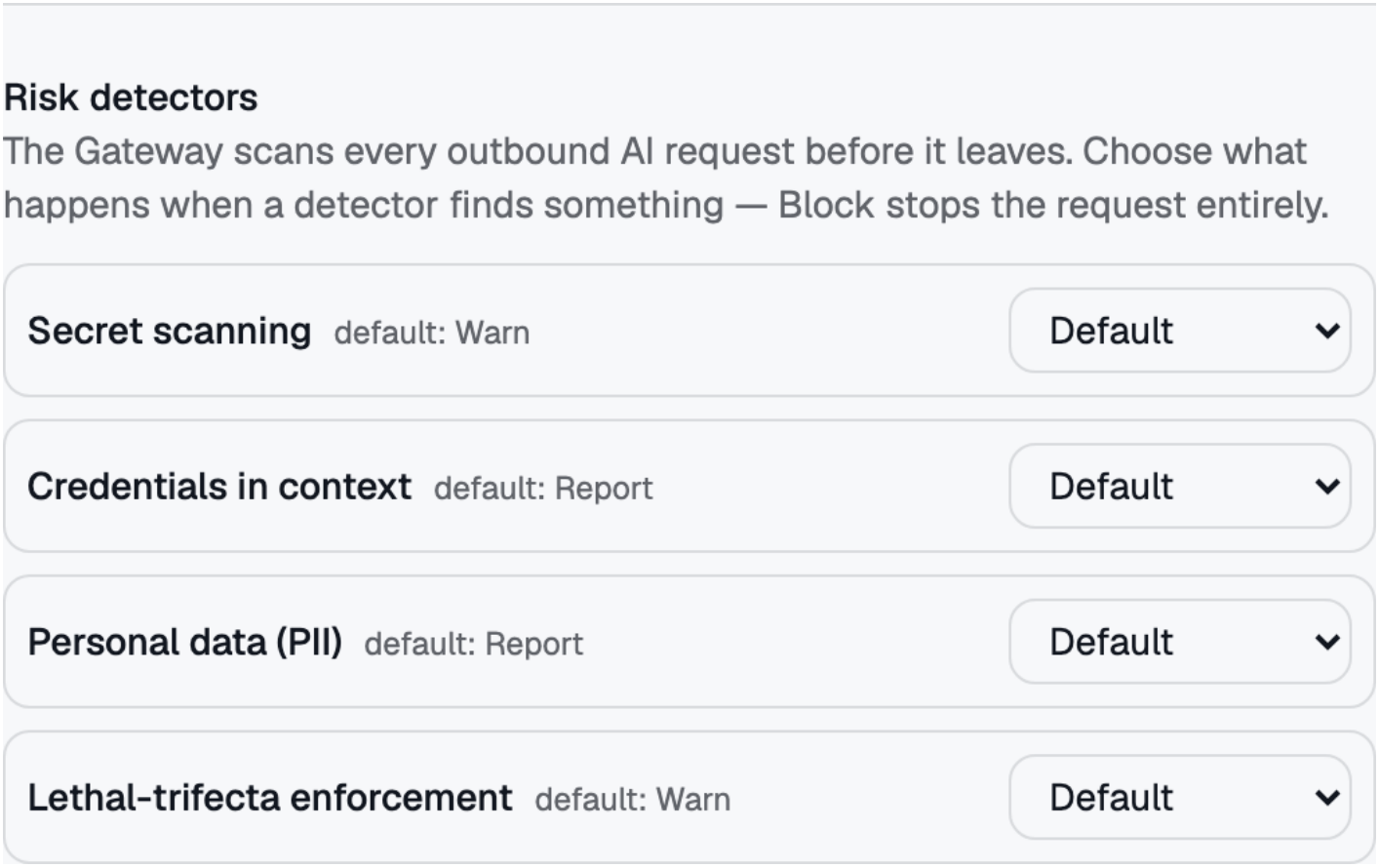
Why you would care

Two reasons, usually. Under GDPR, sending customer personal data to an AI provider is a processing activity you need to have thought about. And separately, most engineering teams simply do not intend to paste production data into a chat window — this is how you find out that it happens.

Tuning

Start at **report**. Look at where the findings cluster: if they come from one team debugging with real data, that is a conversation, not a policy setting. If they are spread evenly, consider **warn** so developers get feedback in the moment.

Block is aggressive for this detector — an email address in a prompt is often completely legitimate — so most organizations stop at warn.



Personal data (PII) is a detector row like any other — choose Warn to coach, Block to enforce.

Lethal-trifecta enforcement

The one detector that is about a *combination* rather than a *thing*. **Default: warn.**

The idea

An AI agent becomes dangerous when three capabilities meet in the same context:

1. **Access to private data** — your repository, your database, your issue tracker.
2. **Exposure to untrusted content** — a web page, an email, an issue written by someone outside your company.
3. **A way to send data out** — an HTTP tool, an email tool, a webhook.

Any one is fine. Any two are usually fine. All three together means a malicious instruction hidden in the untrusted content can direct the agent to read your private data and send it somewhere. The agent is not compromised in any traditional sense; it is doing exactly what it was asked, by the wrong person.

What Sentilai does

Per tool call, per device, the Gateway tracks which of the three legs the device's current context has. When a call would complete the set, the detector's action applies.

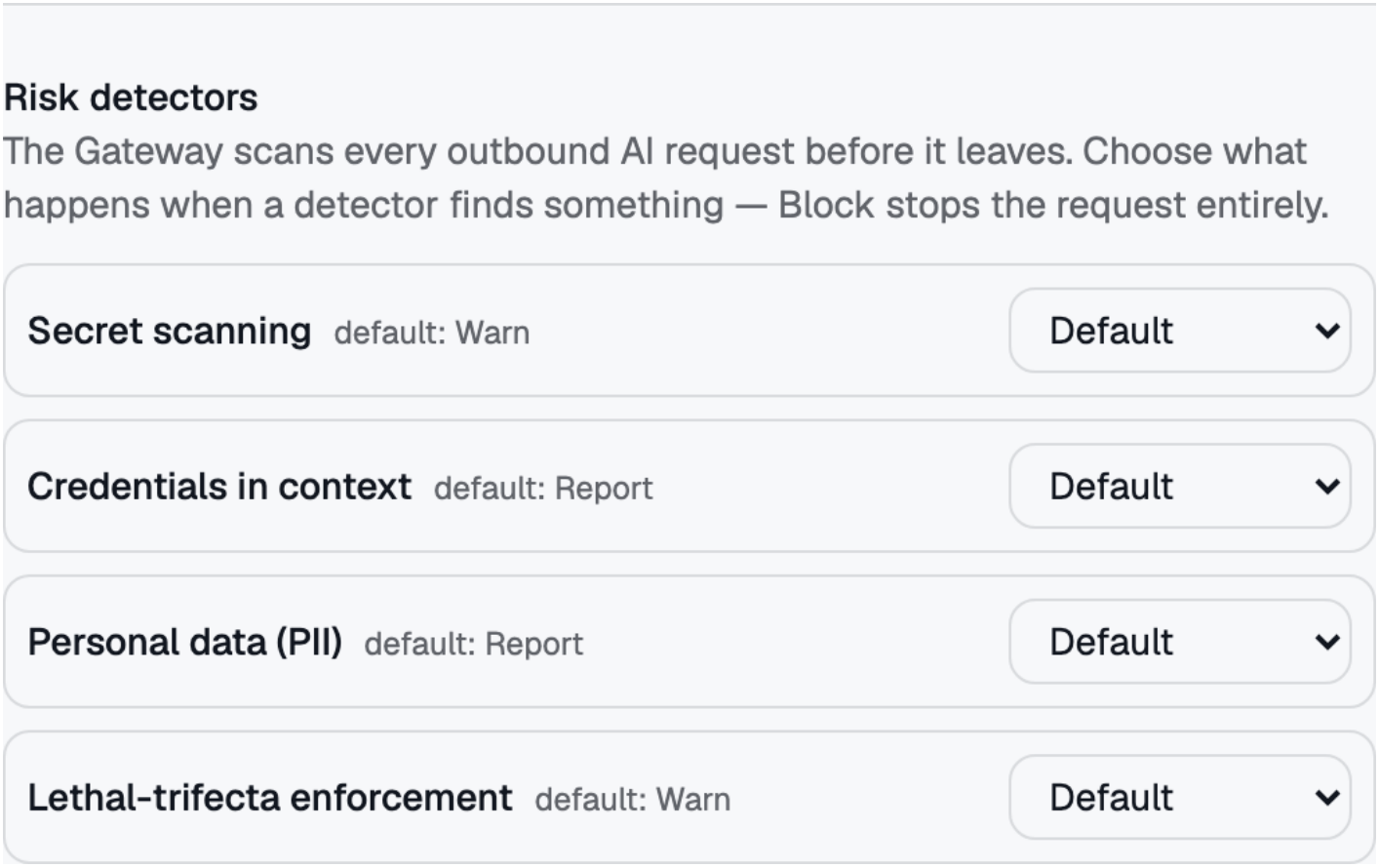
MCP Inventory shows a banner when a device has all three legs present, naming the `server/tool` that contributed each one. That banner is the useful part even in warn mode: it tells you which combination of servers created the exposure, so you can decide which one does not belong.

Where it applies

On the MCP path — the local shim and routed remote endpoints — because that is where tool calls are visible with names. It is not a content scan of the prompt.

Setting it

Policy ? Risk detectors ? Lethal-trifecta enforcement. Warn first; look at the banners; then block once you know which combinations are real.



Lethal-trifecta enforcement in the Risk detectors list — it fires when private data, untrusted content and an egress channel meet in one request.

The prompt-injection classifier

A model, rather than a pattern, judging whether a request is being manipulated.

How it runs

The classifier runs **in parallel with the upstream call**, so it does not add latency to your developers' requests. In Activity you can see how long it took — the millisecond badge in the Signals column — and the score it produced.

The sensitivity dial

Policy ? AI risk classifier sensitivity, three positions:

- **Low** — only the clearest cases.
- **Medium** (recommended) — the default balance.
- **High** — aggressive. At this setting a high-severity verdict can block the next turn of the conversation.

The dial governs this classifier. It does **not** govern secret scanning: detected secrets are handled by that detector's own action regardless of where the dial sits.

Block-next-turn

When the classifier returns a high-severity verdict under High sensitivity, the **conversation** is stopped rather than the individual request — the next turn is refused. This is the right granularity for injection, where the problem is the conversation's context rather than one message.

Blocked conversations are listed at the bottom of the Policy screen with the developer, tool, reason and severity, each with an **Unblock** button. They also clear by themselves when they age out of your retention window.

The paid classifier tier

A second, larger classifier hosted in the EU covers a wider risk taxonomy. It is a disclosed sub-processor — if your DPA review needs the details, ask us and we will send the current list.

Connection modes

Per tool, whether requests run on the developer's own AI subscription or on your organization's provider key.

The two modes

Subscription — the developer's own plan pays. Sentilai governs and audits; billing is unchanged.

Managed API key — the request goes out on a key from **Providers** and is billed to you. One bill, central control, and you can cut off access by removing the key.

Setting it

Policy ? Connection modes, one selector per tool: Claude Code, Cursor, GitHub Copilot, Gemini CLI, Codex CLI.

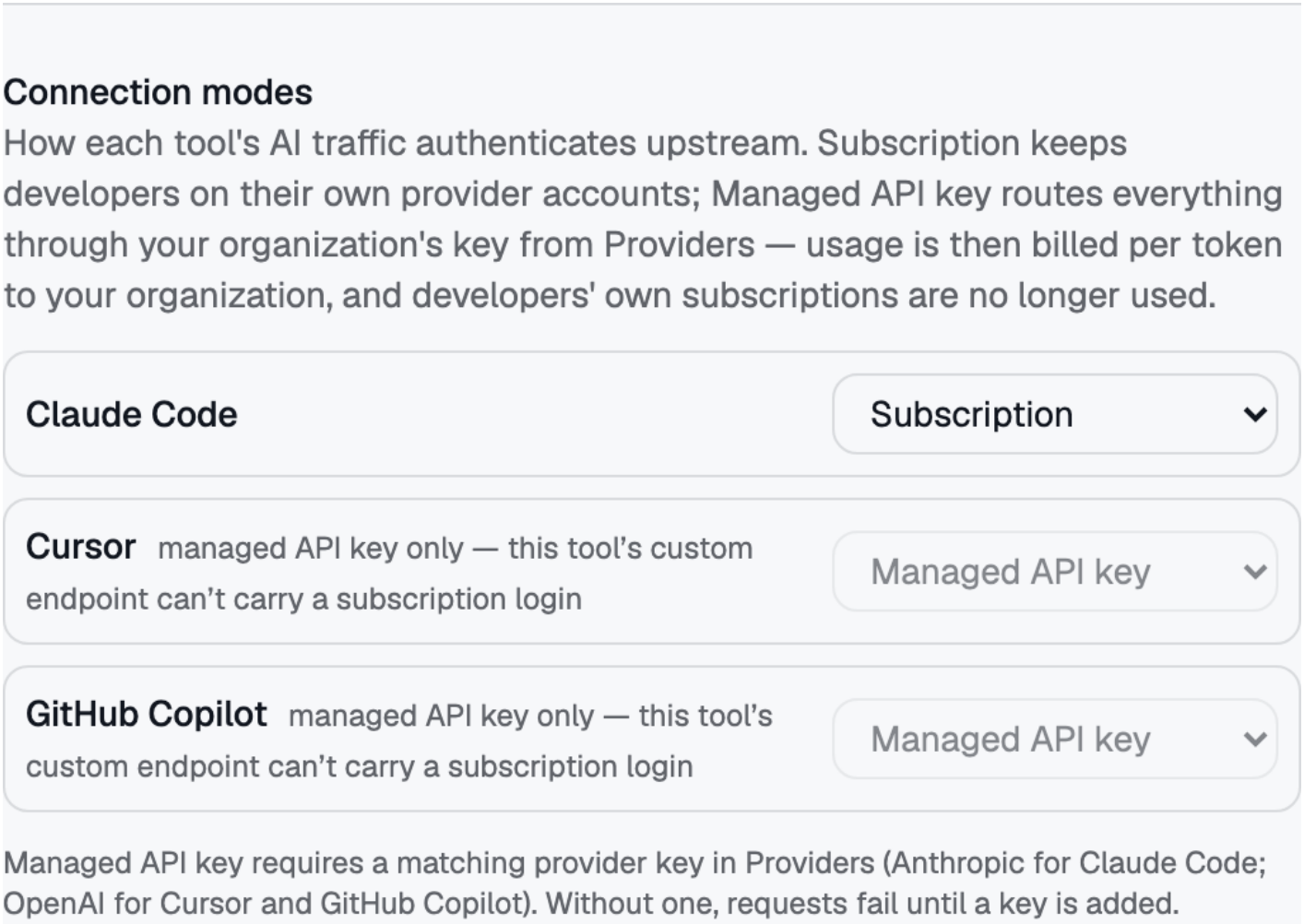
Cursor is locked to managed API key. Its custom-endpoint mechanism cannot carry a subscription login, so governing Cursor requires an OpenAI provider key.

The failure to expect

Managed mode with no matching provider key means requests fail — Anthropic for Claude Code, OpenAI for Cursor and Copilot. The tool shows a provider error and Activity records an upstream error. Add the key, or move that tool back to subscription.

In the audit trail

Activity's **Mode** column shows which mode each request used, and Compliance reports the subscription/API split. Both modes are audited identically; the only difference is who pays.



Connection modes — how each tool's AI traffic authenticates upstream: the developer's own subscription, or a managed key from Providers.

Device sessions

How long a developer's machine may keep working without signing in again.

The two settings

Policy ? Device sessions:

- **Idle timeout (hours)** — a device that has not been used for this long stops working.
- **Max age (days)** — a device stops working this long after enrolment regardless of use.

Leave either blank to switch it off. Both are blank by default.

How quickly it takes effect

Like manual revocation, expiry takes effect within roughly 30 seconds — the Gateway notices at the next token refresh rather than at the moment of expiry.

What the developer sees

Their tools stop working and the Endpoint Suite asks them to sign in again. Nothing is lost; signing in restores the device.

Choosing values

Idle timeout is the one that earns its keep: it quietly retires the laptop of the contractor whose engagement ended, without anyone remembering to do it. A few weeks is usually right — long enough to survive a holiday, short enough to matter.

Max age is a blunter instrument. It is worth setting if you have a compliance requirement that says credentials must be re-established periodically; otherwise idle timeout does the useful part.

Neither is a substitute for offboarding someone who has left. Use **Offboard** for that — it is immediate and it also removes their passkeys.

Local MCP decision cache

Policy ? Local MCP decision cache (seconds), from 0 to 3600.

What it controls

When an AI tool calls a local MCP server, the Sentilai shim on the developer's machine asks the Gateway whether to allow it. This setting is how long the shim may reuse that answer for the same server and tool before asking again.

The trade-off

Higher means fewer round trips and a snappier agent — MCP tool calls happen in tight loops, and a network hop on each one is noticeable.

Lower means a policy change reaches developers' machines faster. At 0, every call asks the Gateway.

A reasonable setting

A few minutes suits most teams. It keeps agents responsive while ensuring that a rule you change at the start of a meeting is in force by the end of it.

If you are actively tuning MCP rules and want to see the effect immediately, drop it to 0 for the session and put it back afterwards.

What it does not affect

Only local stdio MCP servers behind the shim. Routed remote endpoints and the chat-request side-channel are evaluated by the Gateway on every request.

Blocked conversations

When the classifier stops a conversation, it appears at the bottom of the **Policy** screen — and this is where you let it continue.

What you see

Each entry names the developer, the tool, the reason the classifier gave, the severity, and when it happened.

Unblocking

Unblock lets that conversation continue from the next turn. Use it when you have looked at the reason and concluded it was a false positive — for example a developer legitimately working on prompt-injection defences, whose prompts look exactly like the thing being detected.

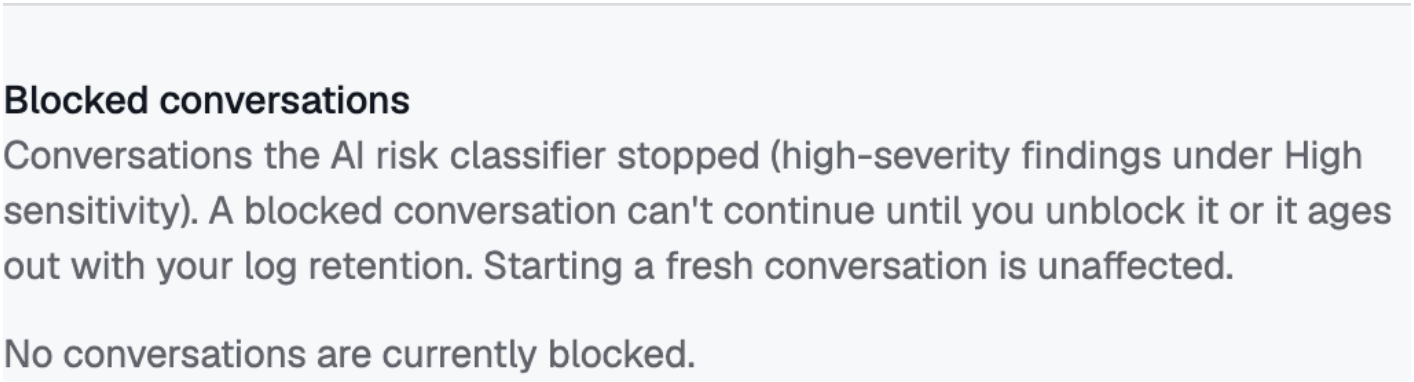
They also expire

Blocked conversations clear by themselves when they age out of your retention window. You do not have to maintain this list.

If it is happening a lot

Frequent blocks usually mean the sensitivity dial is one position too high for your team's work. Move it from High to Medium and see whether the genuine findings survive — they usually do, because the clearest cases are caught at every setting.

An empty list is the normal state and shows "No conversations are currently blocked."



Blocked conversations — sessions the AI risk classifier stopped, and where an admin unblocks one.

Log retention

How long Sentilai keeps your audit events, findings and any captured prompt content.

Where it is

Policy shows it as a read-only card. It is currently set by Sentilai support rather than by you — open a ticket from **Support** and tell us the number of days you need.

What retention covers

Everything time-bounded: audit events behind Activity, risk findings, captured conversations, and the blocked-conversation list. When the window passes, the data is purged rather than archived.

Choosing a number

Two forces pull in opposite directions.

Longer helps you investigate. An incident discovered in June is much easier to understand if you can see April.

Shorter reduces what a breach could expose and what a legal request could reach. If prompt capture is on, this argument gets stronger — captured conversations are the most sensitive thing in the system.

Compliance frameworks rarely mandate a specific number for this kind of telemetry. Pick what you can justify, write down why, and keep the evidence pack from **Compliance** as the record.

If you need longer-term retention

Stream events to your own SIEM — see Stream events to your SIEM. Your retention rules then apply to your copy, independently of Sentilai's window.

Ungoverned-agent policy

On **Policy ? Ungoverned AI agents** you choose what happens when an autonomous AI agent Sentilai does not govern is found on one of your devices. Read Ungoverned AI agents under **Devices and logs** first — this page is only about the consequence.

This is the one setting in Sentilai that can take a developer's AI tools away without an admin doing anything at the time. It is worth ten minutes before you change it.

Three positions

Report it only — the default. The agent appears on the Overview tile and in Diagnostics. The device stays compliant and keeps working. Nothing else happens.

Mark the device non-compliant — the device additionally shows **Non-compliant** in the console. Still nothing is denied. Use this when you want the fact visible in your own reporting without changing anyone's day.

Block the device's access — the Gateway refuses that device's token. Its governed AI tools stop working.

There is deliberately **no "off"**. Detection is not something a tenant switches off; the feature exists so that an admin cannot be unaware. If you do not care, leave it on Report.

The default is Report, and that is not an accident

Nobody is blocked by a Sentilai deployment or upgrade. Blocking is a decision you make on purpose, with a warning on screen telling you what it will do.

What blocking actually does — and does not

Blocking is **coercive, not preventive.**

OpenClaw talks to its provider directly. We are not in that path and cannot stop it. What blocking does is take away the developer's **governed** tools — the traffic that *was* flowing through us, audited and policy-checked — while the ungoverned agent carries on untouched.

The lever is *"you get your AI assistant back when you remove the agent"*, and it works because developers want their assistant. It is not a technical control over the agent, and we will not describe it as one.

If you need to actually stop the agent's own traffic, that is a network-level or MDM control, not this setting.

The grace period

Default **24 hours**. The block applies only once the window has elapsed, measured from **the later of**:

- when the agent was first detected on that device, and
- when you changed the setting.

Both terms matter. Measuring from detection alone would cut off every developer who already had an agent installed the instant you flipped the switch — no warning, nothing they did that day. Measuring from activation alone would give a fresh install after that point no window at all.

Taking the later of the two gives every affected developer the full window from the moment the rule first applies **to them**.

A grace period of **0** is allowed. An admin who wants an immediate block can have one, and the screen says "immediately" when you choose it.

What the developer sees

Not "your credential has been revoked" — that would be false, and would send them looking for an admin who revoked nothing. Their tool reports:

> “ access blocked: an ungoverned AI agent (OpenClaw) was detected on this device — remove it to restore access

It names the agent and the one action that fixes it.

Recovery is automatic

Uninstall the agent ? the device's next poll finds nothing ? the episode closes ? access returns within about half a minute. **No ticket, no admin action, no manual unblock.**

The converse is deliberate: a device that stops polling keeps its open episode and stays blocked. The last thing we actually observed was an agent present, and absence of evidence is not evidence of removal. Since the Endpoint Suite polls on launch, a laptop that comes back from a week off clears itself in seconds.

Timing

Blocks and unblocks propagate on the Gateway's refresh cycle — about 30 seconds, the same as manual device revocation and device-session timeouts.

An explicit revocation still wins

If an admin has separately revoked a device, removing the agent does not restore it. The stronger, admin-initiated fact takes precedence, so removing an agent can never appear to restore access that a real revocation still denies.

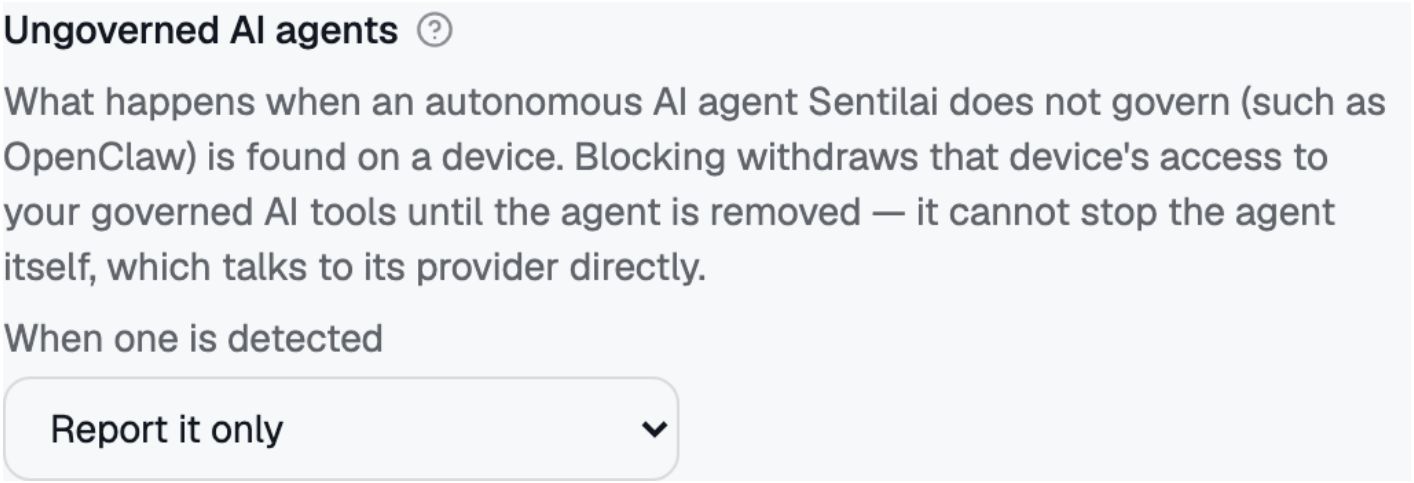
Not available yet

Worth knowing before you plan around them:

- **Per-agent or per-severity policy** — "block on the lethal trifecta, tolerate a sandboxed install". One knob today.
- **Per-developer or per-team exemptions.**
- **An alert when a device gets blocked.** It appears in Diagnostics and in your SIEM stream, but it does not fire a real-time alert channel.

Recommended path

1. Leave it on **Report** and look at what Diagnostics actually shows for a week.
2. If anything appears, talk to those developers before changing the setting.
3. Move to **Block** with the default 24-hour window, not zero.



The ungoverned-agent control: three positions, no "off", and a grace window that only applies to Block.

Credential-store reads

Flags an AI agent reading a file that holds credentials into its context — a `.env`, `~/.aws/credentials`, an MCP configuration with tokens in it, a private SSH key. **Default: warn.**

Why this is separate from secret scanning

Secret scanning catches a key when its *shape* shows up in the traffic. By then the value is already in the model's context. This detector catches the *read itself*: the moment an agent opens the store, whether or not a recognizable key ever appears in what it gets back. It is also what infostealers and misbehaving agents went after first in 2026.

What it looks at

Only two things, both in the agent's own tool calls: the file path a file tool (Read, Edit, Write and their equivalents) was asked to open, and the shell command a terminal tool ran (`cat .env`, `source ~/.aws/credentials`, input redirected from a store). A mention of `.env` in a message is not a read and is ignored. Listing a directory or checking that a file exists is not a read either.

It recognizes a fixed list of stores: `.env` files (but not `.env.example`, `.env.sample` or `.env.template`), Claude Code configuration, MCP configuration files, git credentials, AWS credentials, gcloud credentials, Kubernetes config, `.npmrc`, Docker config, private SSH keys and `.netrc`. The list is in the product, not in your settings — nobody can turn this into a tool for listing files.

What is recorded

The **category** of store and a count — for example `aws_credentials × 1`. Never the path, never the contents. Your developers' directory layout does not end up in the audit record.

Setting it

Warn shows the read in Activity and to the developer. **Block** refuses the request that would carry the file's contents to the model — the honest version of "keep `.env` out of the model's context", which secret scanning alone cannot do. Expect a developer whose agent legitimately needs a local `.env` to notice; the message names

the category so they know why.

When a read is not blocked, it also counts as the *private data* part of the lethal-trifecta check for that device, so untrusted content plus an outbound call afterwards is caught even though the data came through a built-in tool rather than an MCP server.

Project configuration that runs code

Flags repositories whose configuration executes code when an AI agent opens them — a `core.fsmonitor` in `.git/config`, a hook or a `PATH` override in the project's `.claude/settings.json`, an MCP server that starts from `.mcp.json`, a VS Code task that runs on folder open. **Reported, not blocked.**

Why this matters

In 2026 several coding-agent escapes worked the same way: the agent, or a file an attacker got into the repository, wrote configuration that a trusted tool — git, the editor, the agent itself — then ran. Opening the project was enough.

What Sentilai does, and what it does not

The Endpoint Suite checks the projects your developers' agents have opened, on its regular cycle. That means a finding appears **after** the project was opened, not before: this is detection, not prevention. What you get is a record of which repositories carry this kind of configuration and exactly which setting it was — before anyone can clean it up — and the Gateway still governs whatever that code then tries to send.

What is read

Only these files in each project: `.git/config` and the names of executable git hooks, `.claude/settings.json` and `settings.local.json`, `.mcp.json`, `.cursor/mcp.json`, `.codex/config.toml` and `.vscode/tasks.json`. Never your source code, never file contents beyond the setting itself, never environment values — an `env.PATH` override is reported as "env.PATH set", not by its value.

Where you see it

On the device in **Diagnostics**, with the setting and the file it came from. Warning and critical findings also become events in your SIEM export (`project_trust.detected`, and `project_trust.resolved` when a later check no longer finds the setting).