Devices and logs

Device registry, compliance, log collection and sharing.

- Overview
- Devices and revoking access
- Request logs from a device
- Posture findings
- Fake AI application detection
- Revoke and restore access
- Collect logs from a device
- Ungoverned AI agents

Overview

Devices & logs

Device registry

Every machine running the Sentilai Endpoint registers with its own credential (public-key identity). The **Devices** screen lists them with first/last seen, reported hostname, and compliance state. A device credential can be revoked instantly — its requests stop at the Gateway.

Compliance

The app reports whether every installed AI tool on the device is actually governed and whether local MCP servers are wrapped. Non-compliant devices are flagged so configuration drift surfaces.

Event logs

The app keeps a local, allowlisted event log (no prompt content) with fixed 14-day retention. From the console an admin can **request a device's logs**; the app uploads a bundle you can download — useful for troubleshooting a specific machine.

Sharing with support

A log bundle can be explicitly shared with Sentilai support from the console — nothing leaves your tenant without that explicit act, and the share is audited.

Devices and revoking access

Diagnostics lists every machine enrolled in your organization: who it belongs to, when it registered, and when the Gateway last saw it.

Revoking a device

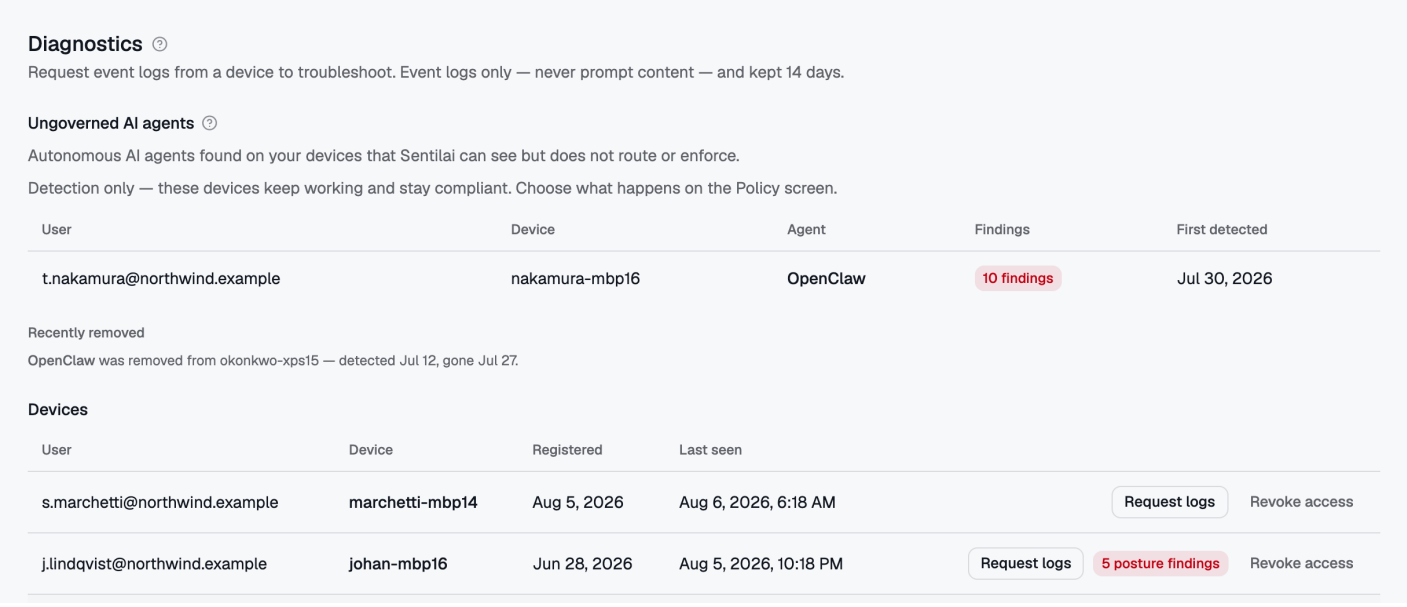
Revoke access on a row kills that machine's gateway tokens. The AI tools on it lose access within seconds — no need to reach the laptop, and it works whether or not the machine is online.

Use it for a lost or stolen laptop, or a machine that shouldn't have been enrolled. For a person who has left, use **Offboard** on Users & Teams instead — it covers every device plus their passkeys in one action.

Revoking is reversible: the same row offers **Restore access** afterwards.

Device identity

Each machine holds a hardware-backed key in the OS keychain (Keychain on macOS, Credential Manager on Windows). The private key never leaves the device, and it is what ties a request in **Activity** to a person and a machine rather than to a shared token.



Diagnostics — enrolled machines, when each was last seen, and per-device revocation.

Request logs from a device

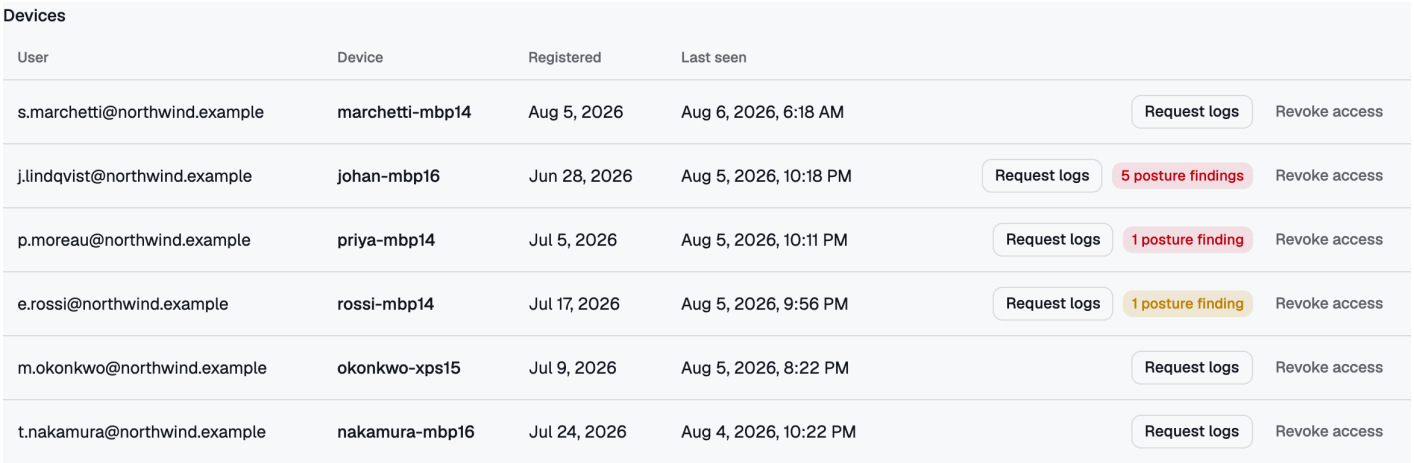
When a developer reports that a tool isn't behaving, you can ask their machine for its event logs without touching the laptop.

1. **Diagnostics** ? **Request logs** on their device row.
2. The Endpoint Suite picks the request up and uploads a log bundle.
3. It appears under **Log bundles**, ready to download.

What is in a bundle

Event logs only — what the app did, which tools it configured, which requests it routed and what happened to them. **Never prompt content.** Bundles are kept 14 days.

If the machine is offline the request waits; the bundle arrives when it next connects.



Each device row on Diagnostics carries Request logs — the bundle appears under Log bundles once the machine uploads it.

Posture findings

The Diagnostics screen can show a **posture findings** badge on a device, coloured by the worst severity found. These are not about Sentilai's configuration — they are about how much autonomy the AI tools on that machine have been given.

Why this exists

An AI coding assistant's own settings decide whether it asks before running a command, whether it can write files freely, and whether it can reach the network from inside its sandbox. A developer who turned all the guard rails off has changed their risk profile far more than any policy you set in Sentilai.

You cannot govern what you cannot see, so the Endpoint Suite reads those settings and reports them.

What is checked

Claude Code

- `permissions.defaultMode` set to `bypassPermissions` — **critical**. The assistant acts without asking, ever.
- The same setting on `auto`, `dontAsk` or `acceptEdits` — **warning**.
- A bare `Bash` or `Bash(*)` allow rule — **critical**. Unrestricted shell.
- Broad `Write`, `Edit` or `WebFetch` grants — worth knowing about.
- Presence of `hooks.PreToolUse` — code that runs before every tool call, which is powerful and worth being aware of.

Codex CLI

- `approval_policy` set to `never` or `on-failure`.
- `sandbox_mode: danger-full-access`.
- `sandbox_workspace_write.network_access: true`.

Per-profile variants of these are checked too.

Report-only

Nothing here is blocked or changed. Clicking the badge lists each finding with its check id, severity, the observed value, and which configuration file it came from — enough to have a specific conversation rather than a vague one.

What is read

Configuration files only. Never prompts, never source code.



A posture badge opens into the findings behind it: Claude Code running with permissions bypassed and a bare Bash grant, Codex CLI approving nothing and sandboxed with full access. Report-only — nothing here is blocked.

Fake AI application detection

On macOS, the Endpoint Suite checks that the AI applications on a machine are actually from the vendors they claim to be.

How

Every macOS application is code-signed with a Team ID. Sentilai compares the Team ID of each installed AI application against the vendor's real, pinned identity — Anthropic, Cursor and Microsoft — and also checks it is installed where it should be.

An application called "Claude" that is signed by somebody else, or that lives somewhere unusual, is flagged.

Why it is worth having

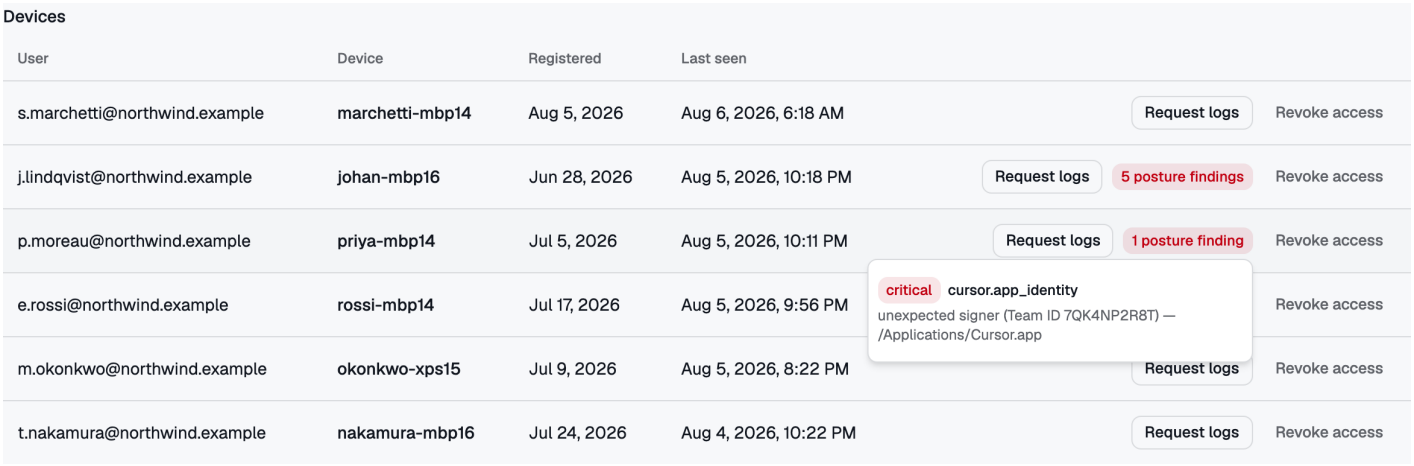
The AI tool on a developer's machine holds a session that reaches your Gateway, and it sees every prompt. A convincing impostor is a very direct route into the middle of your AI traffic. Distributing a lookalike of a popular developer tool is neither novel nor difficult.

macOS only

This depends on the operating system's code-signing infrastructure. There is no equivalent check on Windows yet — which is one of several reasons the Windows track is still behind.

If something is flagged

Treat it as a real finding. Ask the developer where the application came from. The legitimate explanations — a beta from the vendor, a build installed by a package manager — are easy to confirm, and the illegitimate one is worth catching.



The same badge carrying a very different finding: an application installed as Cursor, signed by a Team ID that is not Cursor's. macOS only — it relies on the operating system's code-signing infrastructure.

Revoke and restore access

Cutting a single machine off, without touching the person.

Revoking

Diagnostics ? the device row ? Revoke access. Its Gateway tokens stop working within roughly 30 seconds — online or not, because the cut is made at the Gateway rather than on the device.

Use it for a lost or stolen laptop, a machine you no longer recognize, or a contractor's device at the end of an engagement.

What the developer sees

Their AI tools stop working. The governed configuration remains in place, so the tools **fail closed** rather than quietly reverting to talking to the AI provider directly. This is the entire point: a revoked device must not become an ungoverned one.

Restoring

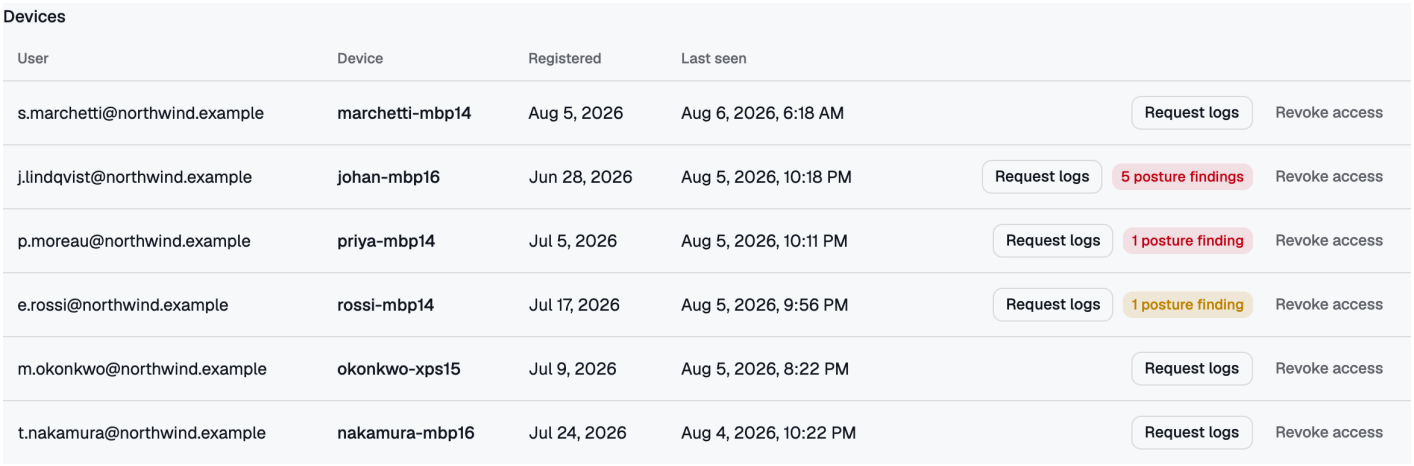
Restore access on the same row puts it back. The device does not need to be re-enrolled.

Revoke or offboard?

- **Revoke a device** when the person stays and the machine goes.
- **Offboard the developer** when the person goes. That revokes *all* their devices, removes their passkeys, and frees their seat — in one action.

Automatic expiry

Policy ? Device sessions can retire devices by idle timeout or maximum age without anyone doing anything. Worth setting: it catches the machines nobody remembers to revoke.



Revoke access sits on every device row — the Gateway notices within about thirty seconds.

Collect logs from a device

When a developer's tools are misbehaving, you can ask their machine for its logs without asking them to find a folder.

Requesting

Diagnostics ? the device row ? Request logs. The button becomes "Logs requested" with a ten-minute cooldown, and the section header counts outstanding requests.

The Endpoint Suite polls roughly every two minutes, so a running machine responds within a few minutes. A machine that is off responds when it comes back.

What arrives

A bundle in the **Log bundles** table: which user and device, its size, when it was uploaded, and its expiry. Bundles are kept for **14 days** and the badge shows the remaining time.

Download fetches it.

What is in it

Event logs only — never prompt content and never secrets. Which tools were detected, what was governed, connection attempts and their errors, update checks, sync activity.

That constraint is what makes this feature safe to use routinely: you can ask any developer for logs without asking them to trust you with what they were working on.

Sharing with support

Share with support attaches a bundle to your organization's support access so we can look at it. This is explicit and per-bundle — support cannot reach into your log bundles otherwise. The row then shows "Shared with support".

Attaching the bundle to a support ticket as well gives us the context to go with it.

Ungoverned AI agents

Most of this documentation is about AI tools Sentilai routes and enforces. This page is about the ones it cannot — and what we do instead.

What an ungoverned agent is

An **ungoverned AI agent** is an autonomous AI system running on a developer's machine that does not send its traffic through the Gateway. It talks to Anthropic, OpenAI or whatever else its own configuration points at, directly. We are not in that path.

We cannot route it, cannot apply your policy to it, and cannot audit what it does. What we *can* do is see that it is there, and tell you.

Today one agent is detected: **OpenClaw**.

Why it matters more than an ordinary unmanaged tool

An AI coding assistant answers a developer's question. An autonomous agent runs a gateway process in the background, takes instructions from a messaging channel, and executes commands on the host. A stock OpenClaw install, with no configuration mistakes by anyone, ships with:

- host command execution set to unrestricted,
- agent sandboxing off,
- and a messaging channel as its interface.

That is private-data access, untrusted input, and a way out — the **lethal trifecta** — all three present by default. Not by misconfiguration. By design.

So the finding is not "somebody set this up badly". The finding is "this exists on a machine that also has your source code on it".

How detection works

The Endpoint Suite scans during its normal poll. Detection is layered so that a config it cannot read still reports the install:

1. **Presence** — the state directory (`~/.openclaw` and any profile variant), the config file, the CLI on `PATH`, `/Applications/OpenClaw.app`, a macOS LaunchAgent, or the Windows scheduled-task script. No parsing at this stage, so an unreadable or hostile config cannot hide the install.
2. **Configuration** — the config file is then parsed for the twelve specific risks below.

Everything read is configuration. Never prompts, never source code, never the agent's conversations.

What is checked

Finding	Severity	Means
Installed	Warning	Presence evidence found. The value names what was seen
Gateway autostart	Warning	Installed as an always-on background service, not started deliberately
Config unreadable	Info	Found, but the config could not be parsed — "we could not look", not "nothing found"
Gateway exposed	Critical / Warning	Listening beyond loopback. Critical when authentication is also absent
Gateway auth none	Critical	Authentication switched off entirely
Tailscale funnel	Critical	Reachable from the public internet
Terminal enabled	Critical	A host terminal served on its control UI
Exec unrestricted	Critical / Warning	Host command execution unrestricted. Critical when set explicitly, Warning when it is the default
Sandbox off	Warning	Agents run without a sandbox — the documented default
Elevated enabled	Critical	Agents can escape the sandbox onto the host
DM policy open	Critical	Anyone who can message the bot can drive an agent with that user's authority
Lethal trifecta	Critical	Host execution, private-data access and an external channel, all at once

The last one is the point of the feature. The others tell you how the machine got there.

Where you see it

Overview carries a tile — "Ungoverned AI agents". When nothing is detected it says so, which is the answer you want most days.

Diagnostics lists each affected device: who, which machine, which agent, how many findings at what severity, and when it was first seen. Below the table, **Recently removed** records agents that were found and are now gone — a real, reassuring answer rather than an empty row.

The full finding list is behind the posture badge on the device row.

Episodes, not snapshots

A detection is recorded as an **episode**: opened the first time the agent is seen, kept open while it remains, closed when a scan no longer finds it. So the history survives — you can tell "never had one" from "had one for three weeks in July".

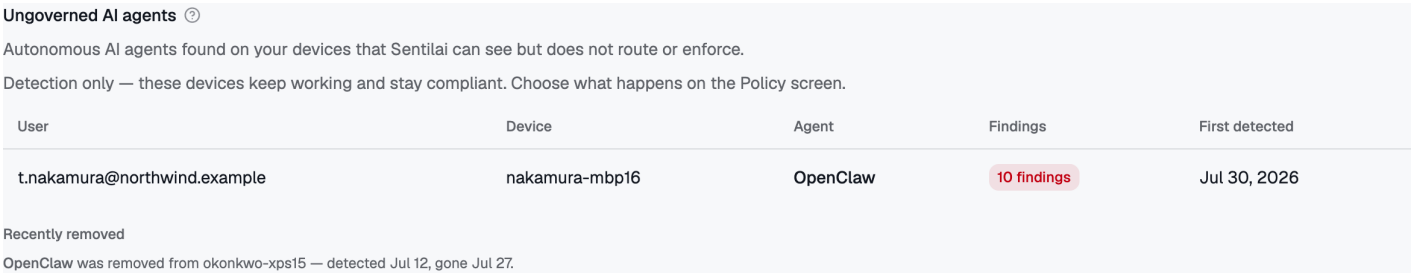
Removal is picked up on the device's next poll. Nothing needs to be cleared by hand.

Honest limits

- **A machine without the Endpoint Suite is invisible.** This finds agents on managed devices. A personal laptop is outside it, the same as everything else.
- **Detection is not interception.** We see the install; we never see its traffic.
- **The list is a list of names we know.** An agent nobody has written a detector for is not detected. OpenClaw is covered because it is the one with real adoption; others follow when they matter.
- **New capabilities can be missed.** A brand-new messaging plugin is not counted as a channel until the detector learns it.

What happens next is your decision

By default, nothing — this is report-only until you say otherwise. See Ungoverned-agent policy under **Policy and risk detection** for the three positions available and what each one actually does.



The ungoverned-agent section: which machine, which agent, how bad, and since when — plus the ones that are gone again.