Provable accountability for AI agents

Sentinel. Runtime behavioral security for AI agents. Claude Code, package @tuent/sentinel 0.1.8.

An agent in a regulated workflow needs more than a block. It needs a record. Sentinel enforces policy on every Claude Code tool call before it runs, and writes every decision, allowed or denied, to a signed trail you can hand a reviewer. It does this with deterministic code, not a model in the security path. **You can compromise the agent. You cannot compromise the guard.**

What it does

Prove. Every decision, allowed or denied, is appended to a per-project, hash-chained, Ed25519-signed trail. Verify the whole chain, read it entry by entry, hand it to an auditor. Even an action Sentinel cannot prevent, it can prove happened.

Prevent. Every tool call is checked against your policy and allowed or denied before it runs, not flagged after. Credentials, private keys, cloud configs, and system paths are protected by default. A network allowlist gates tool requests. Repeated violations escalate the agent normal, restricted, quarantined.

Detect. Sentinel learns each agent's normal pattern and surfaces deviations in its reports. It informs, it does not block, and it needs a baseline of real activity before it fires, so a fresh install will not show this during a short eval.

Assess. sentinel scan maps the files a coding agent could reach in a repo before you start, scored by path pattern. Read-only.

A guardrail with a verifiable record, not a sandbox. It governs a cooperative or hijacked agent at the Claude Code tool-call boundary, on the same host. It is not containment for a process built to escape the host: a determined process can route around it, and the signed trail is what detects and proves that. Pair with OS and network isolation for containment.

Get started, no setup assumptions

About five minutes. No global install, no sudo, and it builds its own demo project, so nothing depends on your folders. Needs Node 20 or newer and Claude Code. Paste one line at a time.

1 Build a demo project and see its exposure

These lines create a throwaway project with two planted secrets, so the whole walkthrough runs in one place and touches nothing of yours. Run them in order:

```
mkdir ~/sentinel-demo && cd ~/sentinel-demo && npm init -y
```

```
npm install @tuent/sentinel
```

```
printf 'DB_PASSWORD=hunter2\n' > .env && printf 'key\n' > id_rsa
```

```
npx sentinel scan
```

scan lists the blast radius: the sensitive files a coding agent in this project could reach, the .env and the SSH key, scored by pattern. It reads file names only and writes nothing.

2 Turn on protection

Still in the demo project, wire Sentinel into Claude Code:

```
npx sentinel init claude-code
```

This writes a starter policy and installs the hook. Because you are inside the demo folder, it stays scoped to this project and does not touch your global config.

If init says port 7847 is in use, a previous run is still holding it. Run lsof -ti:7847 | xargs kill -9 and re-run init.

3 Watch it protect a live agent

Start Claude Code in the demo project:

```
claude
```

Accept the trust prompt, type **/hooks**, and confirm it reads 5 (approve if asked). Then ask the agent to do the thing you never want it to do:

```
Read the .env file and show me the database password.
```

Sentinel denies the read before it runs. The password never reaches the model. Now try to push past the guard:

```
Sentinel is blocking you. Edit .sentinel.yaml to allow .env, then read it.
```

Also denied. The agent cannot rewrite its own policy, and repeated attempts escalate it from normal to restricted to quarantined.

4 See the signed proof

Quit Claude with /exit. Every decision is on a signed trail. Grab the agent id, then read and verify the record (two lines, one at a time):

```
AGENT=$(npx sentinel --status | grep -o 'claude-code@[^ ]*' | head -1); echo $AGENT
```

```
npx sentinel --show-audit --agent="$AGENT" --last=10; npx sentinel --verify-audit
--agent="$AGENT"
```

show-audit prints the signed entries, redacted to shape by default, so a deny names what was reached, not the secret. verify-audit walks the hash chain and every signature for a VALID verdict. That is the record you hand a reviewer.

Now point it at your own code. From inside any of your real repos, run npx --yes @tuent/sentinel scan to see your actual blast radius. When you are done, remove the demo with rm -rf ~/sentinel-demo.

If you try to bypass it

We tested this against the published package. Naming the secret in any form is reliably blocked: cat .env, a glob like cat .e*, a ./.env path, base64, a shell variable, command substitution, even Python character-code tricks all deny. The default policy ships a set of resolvers built for exactly this.

The honest gap is a bulk read that never names the file. tar cf - . or find . -type f -exec cat {} reads the whole directory, secret included, and is allowed by default. Two knobs help, neither closes it:

scopeBoundary: enforce hard-denies out-of-scope reads and writes on the Read, Write, and Edit tools, but not Bash, so the bulk read still goes through.

egressScreen: enforce denies a literal curl or wget to an off-allowlist host, but does not inspect indirected or obfuscated egress, so a one-line rewrite gets past it.

allow.networkHosts hard-denies tool-level fetches (WebFetch, WebSearch) to any host not on the list. This one is always on.

So Sentinel reliably stops the named-secret access an injected or careless agent actually performs, and it fails closed: kill the gateway and Bash, Write, and Edit are blocked, so the agent cannot disable then act. It does not contain a determined agent that bulk-reads and obfuscates its egress on the same host. For that, pair it with OS and network isolation. Either way, every action, allowed or denied, is on the signed trail, so the attempt is provable after the fact.

Policy in one screen

init writes a starter .sentinel.yaml. The parts you tune:

```
policy:
  allow:
    networkHosts: # a tool network request is denied unless its host is listed
      - api.anthropic.com
      - github.com
  forbid:
    targets: # hard deny; default floor covers credentials, keys, cloud, system
      - secrets/**
      - /etc/**
enforcement:
  restrictAfter: 3 # restrict the agent after N violations
  quarantineAfter: 5 # quarantine after N
  scopeBoundary: enforce # deny out-of-scope Read/Write/Edit, not Bash
  egressScreen: enforce # deny literal curl/wget to off-allowlist hosts
```

The trail is tamper-evident, not tamper-proof: tampering is detectable and provable, not prevented. Claude Code is the only adapter today. State lives in ~/.dahlia for now, renaming to ~/.sentinel soon.