

Runveil

A firewall for what your AI tools send

Real-time DLP for the AI era — catch secrets before they leave the machine

The problem

- Developers send code, prompts, and files to AI tools **all day**
- Those payloads carry **secrets** — API keys, tokens, private keys, DB URLs
- Once it's sent, **it's gone** — logged on someone else's servers
- Existing DLP can't see inside TLS to `api.openai.com` / `api.anthropic.com`

| AI egress is a security blind spot.

It just got worse

- AI **coding agents** (Claude Code, Cursor) now **read your files and send them automatically** — you don't even paste
- One "summarise this repo" can ship a `.env` to the cloud
- Secrets sprawl at **machine speed**, no human in the loop

The solution: Runveil

A lightweight **local proxy** that inspects every AI request in real time and enforces a **policy**:

allow · **warn** · **redact** · **block**

- Secrets caught **before** they leave the machine
- **redact** masks the secret and **lets the tool keep working**
- Every request audited to a central **control plane**

● LIVE DEMO — Act 1: the developer's machine

1. `claude` running through Runveil; repo has a file with an AWS key
2. Ask: *"read leak.txt and summarise"*
3. → **HTTP 403 — blocked by Runveil**. The key never reached Anthropic
4. Flip policy `block` → `redact` (hot-reload) → **Claude works**, key **masked**

● LIVE DEMO — Act 2: the control plane

Switch to the **dashboard** (the team / SaaS view):

1. **Live overview** — the request appears in the fleet stream, blocked one in red
2. Click → **Request detail** — `aws_access_key_id · high · rule block-high-secrets`, which device, when
3. **Analytics** — requests, errors, latency, **count by device & host**
4. Policy **distributed from here** — one change rolls out to every device

How it works: two halves

```
AI tool (Claude Code / Cursor / curl)
  | HTTPS_PROXY
  v
Runveil AGENT -- local CA, MITM --> policy engine (31 detectors)
  (open source)                    allow / warn / redact / block
  | audit (metadata + findings, no bodies)
  v
CONTROL PLANE --> central policy . fleet audit . dashboard
(the SaaS)
```

- **Agent (OSS):** scans + enforces locally; content never leaves the machine except the masked request
- **Control plane (SaaS):** pushes policy, ingests audit — **findings only, never prompts**

The policy engine

- **Rules, first-match-wins** — match by `severity`, detector `pattern` (glob), or file `path`
- **4 actions:** allow · warn · redact · block
- **31 built-in detectors:** AWS, GitHub, OpenAI, Anthropic, Stripe, Slack, Google, private keys, DB URLs, JWTs...
- **Hot-reload + central distribution**

```
rules:  
- name: never-leak-private-keys  
  match: { pattern: private_key_* }  
  action: block  
- name: redact-other-secrets  
  match: { severity: high }  
  action: redact
```

The control plane is the product

Open core → SaaS

Today — built & demoed	SaaS roadmap
Fleet live stream, analytics, device + request detail	Multi-tenant, SSO / RBAC
Central policy distribution	Policy authoring + approvals in-UI
Audit ingest (metadata + findings)	Retention, SIEM export
Per-device id, org model	Compliance reports (SOC2)

Free agent = adoption. Paid control plane = revenue.

Why Runveil wins

- **Inline & real-time** — prevents the leak, not a post-mortem log
- **Redact = DLP that doesn't break workflows** — the killer feature; block-only tools get turned off
- **Per-device policy + audit** — central control, local enforcement
- **Open source** — drop-in proxy, no vendor lock-in

Built and working today

Agent · 31-detector policy engine · Control plane · Dashboard

Demoed live, end-to-end.

Status & the ask

Working today:

- Agent (proxy + 31-detector engine + CA) — open source
- Control plane (policy distribution + audit + full dashboard)
- End-to-end **demoed live**: block/redact → audit → dashboard
- Docs (policy guide)

The ask: pilot design partners · feedback on the policy model · the prize 🏆

Thank you

Runveil — stop secrets from leaking into AI

github.com/dawnn07/runveil