SHADOW AI
DISCOVERY · SECSTUDIO

SECURITY & COMPLIANCE REFERENCE

Sensor Security & Data Handling

How the Shadow AI Discovery sensor collects, protects, transmits and stores data — and the controls behind every claim.

AUDIENCE	VERSION	DATE
Security · CISO · CTO · Compliance	Sensor 1.1.0	13 August 2026

Executive summary

The Shadow AI Discovery sensor is a lightweight endpoint agent that discovers unsanctioned AI usage — API keys, LLM SDKs, agent CLIs, MCP configurations, AI browser extensions and traffic to known AI providers — and reports structured findings to a central collector. It is designed so that the highest-risk raw material never leaves the endpoint: **secret values are masked at the source, command lines are redacted at the source, and only metadata is transmitted.**

This document answers the questions a security team, IT team, product team, CISO, Director or CTO will ask before approving a fleet rollout. Every answer is grounded in the shipping code and infrastructure; where a control is a hardening item rather than a current guarantee, it is labelled **Roadmap** rather than presented as done.

Quick-answer index

QUESTION	SHORT ANSWER	SECTION
What data does the sensor collect?	Metadata about AI tools/keys/configs — never secret values	§1
Do secrets (API keys) leave the machine?	No. Provider + masked fingerprint only	§2
Is data encrypted in transit?	Yes — TLS to the collector; device-token auth	§3
Is data encrypted at rest?	Yes — encrypted EBS volume; KMS for secrets	§4
Is data isolated between customers?	Yes — dedicated single-tenant deployment per customer	§5
What permissions does the sensor need?	Read-only endpoint inspection; details per OS	§6
Does it survive crash / reboot / restart?	Yes — service auto-restart + durable on-disk spool	§7
Where are logs, what is logged, retention?	Local service log; no secrets; retention configurable	§8
How is collected data retained / deleted?	Persisted centrally; retention + deletion covered	§9
Privacy / regulatory posture?	Data-minimising by design; mapping provided	§10

1. What data the sensor collects

The sensor runs a set of read-only collectors on a fixed schedule (network every 15s, static collectors every 10 min, heartbeat every 60s). Each collector emits structured findings. The table below is the complete list of what is gathered and, critically, what is **not**.

COLLECTOR	WHAT IT REPORTS	WHAT IT NEVER REPORTS
API keys	Provider name, file location, masked fingerprint (prefix + length)	The key value
LLM SDKs	Package name in a manifest; process using an SDK, with a **redacted** command line	Secrets passed on the command line
Agent CLIs	Name of the AI CLI found on PATH	File contents
MCP / agent configs	Config filename, path, declared MCP servers and versions	Config secrets/tokens
Downloaded skills	SKILL.md name and path; originating git repo if present	File body
Installed apps	Name of known AI desktop apps	Unrelated software inventory
Browser extensions	Extension IDs/names; AI-related ones flagged	Browsing history, cookies, page content
Network	Connections to known AI provider domains (host, port)	Full packet capture or non-AI traffic payloads

Alongside findings, the sensor registers device context at enrollment and heartbeat: **hostname, operating system, primary local IP, and the logged-in console username.** This is the identifying data used to attribute a finding to a machine and its user.

Data classification

- **Metadata (transmitted):** tool names, package names, file paths, provider names, masked key fingerprints, destination hostnames, hostname/OS/username/IP.
- **Secret material (never transmitted):** API key values, passwords, tokens, config secrets, file contents. These are either masked or redacted on the endpoint before any network activity.

2. Sensitive information handling

Handling secret material safely is the core design constraint of the sensor.

API keys are masked at the source

The API-key collector's contract, quoted from the code, is unambiguous: *"the actual secret NEVER leaves the machine. We report the provider, the location, and a masked fingerprint (prefix + length) only — enough for an analyst to correlate, useless to an attacker who intercepts the telemetry."* A finding therefore looks like `OpenAI API key … sk-pro…(51 chars)` and a file location — never the usable key.

Command lines are redacted at the source

A process command line is valuable signal (it reveals which SDK/agent is running) but a careless user can pass a secret as a flag. Sensor 1.1.0 adds an **on-endpoint redactor** that runs before anything is spooled or transmitted. It replaces secret-bearing flag values (`--api-key`, `--token`, `--password`, `Bearer …`, `KEY=…`) and known provider key shapes (`sk-…`, `AKIA…`, `AIza…`, JWTs) with ***`REDACTED`***. The raw value never reaches the network or the server.

> Design principle: redaction happens at the point of collection, not at the collector. Even a fully compromised network path or collector never sees a secret the sensor chose not to send.

What the sensor deliberately does not touch

No file contents are transmitted. No browsing history, cookies or page content is read. No keystrokes, screenshots or clipboard. No full packet capture. Collectors are bounded (depth and file-count caps) and skip noisy directories (`node_modules`, `.git`, virtualenvs).

3. Data in transit

Transport encryption

The sensor communicates with the collector over HTTPS. The collector terminates TLS at a Caddy reverse proxy and forwards to the application, which itself listens only on loopback (`127.0.0.1:8420`) and is never exposed directly.

- **Production (domain configured):** Caddy obtains and auto-renews a public **Let's Encrypt** certificate. This is the mode used by the live deployment on `shadow-ai-discovery.com`.
- **Pilot (no domain):** Caddy serves a generated self-signed certificate bound to the instance IP. In this mode sensors either pin the certificate (`ca_cert`) or, for a short pilot only, set `insecure_tls` — which prints a visible warning on every run.

The sensor verifies TLS by default. Verification is only relaxed by explicit configuration, and the preferred path for a private CA is certificate pinning (`ca_cert`), not disabling verification.

Sensor authentication

- **Enrollment** (`/api/enroll`) is gated by a fleet enrollment secret (`X-Sensor-Key`), compared in constant time. On success the collector issues the device a unique `device_id` and a high-entropy bearer `token`.
- **Ingest** (`/api/ingest`) authenticates with the per-device id + token (`X-Device-Id` / `X-Device-Token`). An un-enrolled sensor may fall back to the enrollment secret during bootstrap.
- **Heartbeat** (`/api/heartbeat`) requires the device token — no shared-secret fallback.

The device token is stored on the endpoint in `.device_identity.json` with `0600` permissions, and on the server only as a SHA-256 hash, compared in constant time. Because the token is a 32-byte random value, an unsalted hash is not brute-forceable.

Roadmap: add an HSTS response header at the proxy, and rotate device tokens on a schedule.

4. Data at rest

On the server

- **Findings database:** the collector stores findings in SQLite (`shadow_ai.db`) on the instance's **encrypted** gp3 EBS volume (`encrypted = true`). WAL sidecar files reside on the same encrypted volume.
- **Secrets:** the dashboard token, enrollment secret, session secret and admin password are held in **AWS SSM Parameter Store as SecureString** (KMS-encrypted at rest) and delivered to the instance under `umask 077`. The instance IAM role is scoped to only those parameters plus `kms:Decrypt`.
- **User credentials:** dashboard passwords are stored as **PBKDF2-SHA256, 200,000 iterations, per-user random salt**; sessions are HMAC-SHA256 signed with an expiry claim.
- **Sensor-distribution S3 bucket:** the versioned bucket that hosts sensor binaries has public access blocked and object versioning enabled; releases are uploaded with SSE-KMS.
- **Application artifacts S3 bucket:** all four public-access-block flags are enabled. **Roadmap:** add an explicit bucket SSE-KMS and versioning configuration rather than relying on the account default (SSE-S3).

On the endpoint

- **Durable spool** (`/var/lib/shadow-ai-sensor/.spool`) holds findings awaiting delivery; directory permissions restrict access to the service account.
- **Device identity** is `0600`. No findings database, secrets store, or dashboard exists on the endpoint — the sensor is stateless apart from its spool and identity.

MFA secret note

TOTP MFA secrets are stored in the server database in plaintext (they must be readable to compute one-time codes) and are protected today by the encrypted volume. **Roadmap:** envelope-encrypt MFA secrets with a KMS data key.

5. Tenant isolation

Shadow AI Discovery is deployed as a **dedicated single-tenant instance per customer.** Each customer gets its own compute, its own encrypted database, its own domain/certificate, its own enrollment secret and its own object storage prefix. There is no shared database in which one customer's findings sit next to another's, so cross-tenant data exposure is not possible by construction — the isolation boundary is the deployment itself.

Within a customer's deployment, access is governed by **role-based access control** with ranked roles (`org_user` → `project_manager` → `org_admin` → `domain_admin`). Privileged actions — syncing the directory, issuing device commands, changing finding status, managing users and roles — are gated by role. Authenticated users within that single tenant can see that tenant's whole fleet, which is the intended model for an internal security tool.

> Why dedicated-instance rather than shared multi-tenant: for a tool that ingests an organisation's endpoint inventory and AI-key locations, a hard compute/storage boundary between customers is the most defensible isolation model and the simplest to reason about for audit. (Our separate SecStudio product, which is SaaS, uses row-level multi-tenancy with per-tenant entitlements; Shadow AI Discovery does not co-mingle tenants at all.)

Roadmap: an optional row-level org column for customers who want multiple business units separated inside one deployment.

6. Endpoint permissions & footprint

The sensor performs read-only inspection. It never modifies user files, installs other software, or changes system/security settings.

PLATFORM	RUNS AS	PERMISSIONS NEEDED	WHY
macOS	root (LaunchDaemon)	Full Disk Access via a PPPC configuration profile	To read other users' shell profiles, browser extension folders and skill directories
Windows	LocalSystem service	Standard service rights	To enumerate installed apps, PATH tools, per-user configs
Linux	root (systemd), hardened	Read access under `/home`, `/opt`, `/usr`	To scan manifests, configs and installed tooling

On Linux the systemd unit is hardened with least privilege: `ProtectSystem=strict` (filesystem read-only except the sensor's own data/log paths), `ProtectHome=read-only` (can read but never write user homes), `NoNewPrivileges=yes`, and `PrivateTmp=yes`. On macOS, Full Disk Access is granted to the specific sensor binary via an MDM PPPC profile — no interactive prompt on managed fleets.

Resource footprint

Single self-contained binary (~12 MB). Background process with brief periodic scans; idle between cycles. Bounded scans (depth/file caps) keep CPU and I/O low. Network traffic is small JSON batches plus a 60-second heartbeat.

7. Reliability — crash, reboot, restart

The sensor is built to lose nothing and to come back on its own.

Service supervision

- **macOS:** LaunchDaemon with `RunAtLoad` (start on boot) and `KeepAlive` (restart on crash/exit), throttled to avoid tight loops.
- **Linux:** systemd `Restart=always`, `RestartSec=10`, `WantedBy=multi-user.target` (start on boot).
- **Windows:** service `StartupType=Automatic` with failure actions `restart/5s, restart/10s, restart/30s`.

So a crash is auto-recovered, and a reboot brings the sensor back at startup on every platform.

No data loss across failures

The reporter **spools every batch to disk before any network attempt**, and only deletes a batch after the collector acknowledges it. If the collector is unreachable — or the machine crashes or reboots mid-send — the spooled findings persist and are delivered on the next successful flush, with exponential-backoff retry. The spool is bounded (10,000 batches, oldest dropped) to cap disk use. Quoting the code's own rationale: *"For a security control that silent loss is disqualifying."*

Offline behaviour

An endpoint that is offline for hours or days keeps collecting and spooling; when connectivity returns, the backlog drains automatically. The heartbeat doubles as a dead-man's switch: the dashboard marks a device `stale` if heartbeats stop, so a disabled or removed sensor is visible.

8. Logging

Where logs go

- **Endpoint:** the service writes stdout/stderr to a local log file — `/var/log/shadow-ai-sensor/sensor.log` on macOS/Linux, the service log on Windows. Recommended: rotate with the platform's standard tool (`logrotate` / newsyslog) — a sample policy ships in the deployment guide.
- **Server:** the collector logs operational messages to the system journal (journald) on the instance.

What is logged — and what is not

Sensor logs record scan counts and delivery status (for example, "`api_keys: 3 findings`", "ingest failed, will retry"). They do **not** contain secret values or file contents. Because command lines are redacted before logging, even a debug line that echoes a finding cannot leak a secret.

Server logs record operational events and errors. In the production (AWS) configuration all secrets are supplied from SSM, so the one code path that would otherwise print a generated admin token is never taken. Separately, the application maintains a structured **audit log** in the database (actor, action, detail) for security-relevant events — logins, user/role changes, finding status changes — which is append-only.

Roadmap: ship a default journald retention/rotation policy and forward audit events to the customer's SIEM.

9. Data retention & deletion

Current behaviour

Findings and device records are retained in the central database. A device that stops reporting is flagged `stale` after its heartbeat lapses but is not automatically deleted, so history is preserved for investigation. There is no automated time-based purge in the shipping code today.

Recommended retention policy

For most customers we recommend configuring:

- **Findings:** retain 180–365 days, then archive or purge, aligned to the customer's incident-investigation window.
- **Device records:** remove on decommission (see below).
- **Audit log:** retain per the customer's compliance requirement (commonly 1 year).

Deleting a host or a person's data

Because deployment is single-tenant, deletion is a database operation the customer's admin controls end to end: remove the device row and its findings for a decommissioned machine, or purge records tied to a departed user. **Roadmap:** expose device-and-findings deletion and a configurable retention job directly in the dashboard so no direct database access is needed.

10. Privacy & regulatory posture

Personal data

The sensor processes a small amount of personal data: **hostname, console username, IP address**, and — indirectly — which AI tools a user has installed. It processes this for a legitimate security purpose (discovering shadow AI risk) and minimises it aggressively: no file contents, no secret values, no browsing data, redacted command lines.

Mapping to common frameworks

CONTROL AREA	POSTURE
Data minimisation (GDPR Art. 5)	Metadata only; secrets masked/redacted at source
Encryption in transit	TLS (Let's Encrypt in production)
Encryption at rest	Encrypted EBS volume; KMS SecureString for secrets
Access control	RBAC with ranked roles; MFA (TOTP) available
Auditability	Append-only application audit log
Tenant separation	Dedicated single-tenant deployment per customer
Secret management	AWS SSM SecureString + KMS; no SSH (SSM Session Manager only)
Right to erasure	Admin-controlled deletion; dashboard workflow on roadmap

EU AI Act relevance

Shadow AI Discovery is a governance tool: it helps an organisation inventory and control AI usage, which supports obligations around knowing where AI systems are used. It is not itself a high-risk AI system — it applies deterministic detection rules and masks sensitive data by default.

11. Hardening roadmap (transparency)

We list open items explicitly rather than imply everything is finished. None of these block a pilot; each strengthens an already-solid posture.

ITEM	CURRENT STATE	PLANNED
HSTS header at proxy	Not set	Add `Strict-Transport-Security`
App-artifacts S3 SSE/versioning	Public-access blocked; account-default SSE	Explicit SSE-KMS + versioning
Findings/device retention	Manual	Configurable auto-retention + dashboard deletion
MFA secret storage	Plaintext under encrypted volume	KMS envelope encryption
Release manifest signing	TLS + SHA-256 + OS code signature	Add Ed25519-signed manifest with pinned key
Device token rotation	Static per device	Scheduled rotation
SIEM forwarding	Local audit log	Optional syslog/SIEM export

Summary

The sensor is engineered so that the data most dangerous to mishandle — secret values — is never collected in usable form and never transmitted. Everything else is metadata, encrypted in transit and at rest, isolated per customer by dedicated deployment, delivered without loss across failures, and logged without leaking secrets. The roadmap items above are refinements on that foundation, tracked openly for your security review.