AiSOC Technical Guide

Install, operate and integrate — on a workstation and on a server

AiSOC project · v1.5 · Released 2026-11-05

MIT licensed · github.com/beenuar/AiSOC

Who this is for

You are installing AiSOC, or you have installed it and something is not behaving. The guide is linear — each section assumes the one before it — but the troubleshooting tables are written to be read on their own at the moment something breaks.

Three audiences, and the paths diverge early:

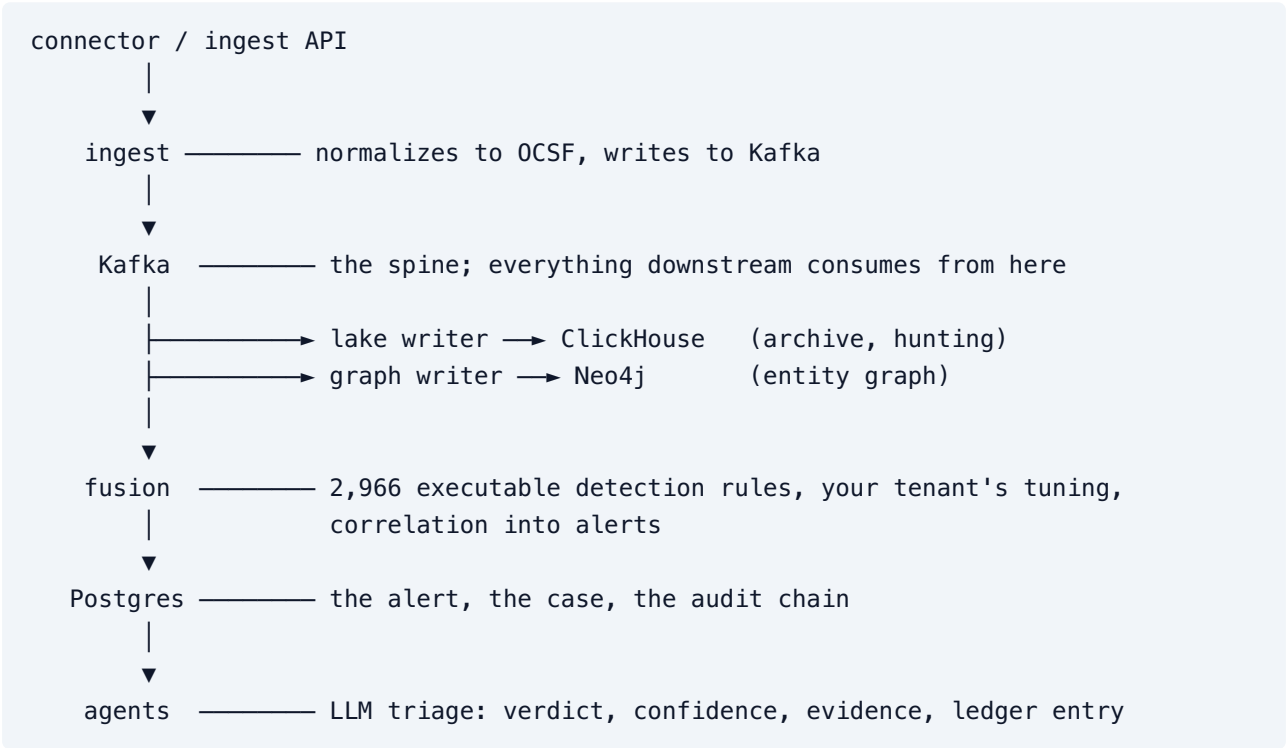
You are	Start at	Time
Evaluating on a laptop	Workstation install	~10 minutes
Deploying for a team	Server install	~45 minutes
Integrating with something	The REST API or MCP	—

---

1. What you are installing

AiSOC is a security operations platform: it ingests security telemetry, normalizes it, runs detections, correlates what fires into alerts, and has an AI agent triage them. It is not a SIEM replacement — it does not do long-term log retention or compliance search — and it does not discover telemetry you have not connected.

The pipeline, end to end



Deployment profiles

Profile	Services	Memory	Disk	What you get
CORE (`make up`)	16	8 GB	20 GB	The whole pipeline, AI triage with a local model, real threat intel. No credentials of any kind
full (`make up-full`)	22	12 GB	40 GB	CORE plus ClickHouse, Neo4j, Qdrant, UEBA — hunting, the entity graph, behavioural analytics

CORE is not a demo. It runs the real pipeline and a real local model; what it leaves out is the stores that only matter once you have volume.

Where the model runs

A model ships with AiSOC and runs on CPU, which is why `make up` needs no account, no key and no GPU. It is also the slowest of four options.

Option	Command	Works on
Bundled model, CPU	`make up`	everything
Bundled model, NVIDIA GPU	`make up-gpu`	Linux, Windows + WSL2
An Ollama you already run	`make up-host-llm`	everything, and the **only** GPU route on a Mac
A hosted provider	configured in the console	everything

`make doctor` tells you which fits the host you are on. The first-run wizard's **Choose where the AI runs** step tells you which is in effect right now.

The GPU reservation is an overlay, not a line in the base compose file, and that is deliberate rather than tidy. A host without an NVIDIA GPU and the container toolkit cannot start a service that reserves one — the daemon answers `could not select device driver "nvidia"` and refuses — so putting it in `docker-compose.yml` would take out first run for every Mac, every CPU-only Linux box and every CI runner.

`make up-gpu` runs `scripts/check_gpu_runtime.py` first, which exists because the daemon's own error names neither a cause nor a next command. The preflight distinguishes a missing card, driver, toolkit and daemon configuration, and names the install step for whichever is absent.

On Apple Silicon, `make up-gpu` cannot help and the preflight says so. Docker Desktop does not pass the Metal GPU into a Linux container — no toolkit, driver or setting changes that, so a container on a Mac is CPU-only whatever is reserved. A natively-installed Ollama does use Metal:

```
brew install ollama
OLLAMA_HOST=0.0.0.0 ollama serve          # in its own terminal
ollama pull llama3.2:3b-instruct-q4_K_M
make up-host-llm
```

`OLLAMA_HOST=0.0.0.0` matters: Ollama binds loopback by default, which a container cannot reach.

`make up-host-llm` is worth running even without a GPU. `11434` is in the managed port inventory, so on a host already running Ollama, `make up` sees the conflict and republishes the bundled one on a free port — leaving you with two Ollamas, the stack talking to the new one, and yours idle.

Checking what is actually running

A GPU reservation is a **request**. A model can still land on the CPU for want of VRAM or a usable driver, so reading the compose file back would report the intent and call it the outcome. Ask instead:

```
curl -s localhost:8000/api/v1/llm/runtime -H "Authorization: Bearer $TOKEN"
docker compose exec ollama ollama ps     # SIZE and the GPU/CPU split
```

Not loaded right now is one of the answers: Ollama unloads after a few minutes idle, so an empty answer means nobody has asked it anything yet. That is not the same as CPU, and reporting CPU there would be a guess about the exact thing you are deciding on. It is also not the same as having no model at all, which is why `no_model` is its own answer — a first run whose one-time download failed needs `docker compose run --rm ollama-pull`, not a triage.

`placement`	Means
`gpu`	the whole model is in VRAM
`partial`	split; `detail` says what fraction, which explains the latency
`cpu`	entirely on the CPU
`no_model`	Ollama holds no model at all — the one-time download never completed
`unknown`	a model exists but none is loaded, so it cannot say where one would run
`unreachable`	nothing answered — expected on a hosted-provider deployment

---

2. Workstation install

For evaluating, developing, or running a small deployment on a laptop.

Prerequisites

Requirement	Minimum	Why
Docker	24+ with Compose v2	The whole stack is containers
Memory allocated to Docker	8 GB for CORE, 12 GB for full	Below this, services restart-loop
Disk free in the Docker VM	20 GB	Kafka corrupts its log directory when it runs out, and its healthcheck still passes
`make`, `git`, `curl`	any	Driving the stack

On macOS the Docker VM has its own disk allocation, separate from your machine's. Docker Desktop → Settings → Resources is where you raise it, and "I have 500 GB free" is not the relevant number.

Install

```
git clone https://github.com/beenuar/AiSOC.git
cd AiSOC
make up
```

That is the whole thing. `make up` generates real secrets into `.env`, creates an administrator, and prints the password once.

Do not run `cp .env.example .env` first. It used to be a documented step and it made things worse: the example file carried an invalid Fernet placeholder, and the credential vault treats empty as "use an ephemeral dev key" while non-empty-and-invalid is fatal. Copying the file gave you HTTP 500 on every connector save, discovered at the connector wizard rather than at boot. Not copying it gave you a working vault.

Verify it came up

```
make doctor
```

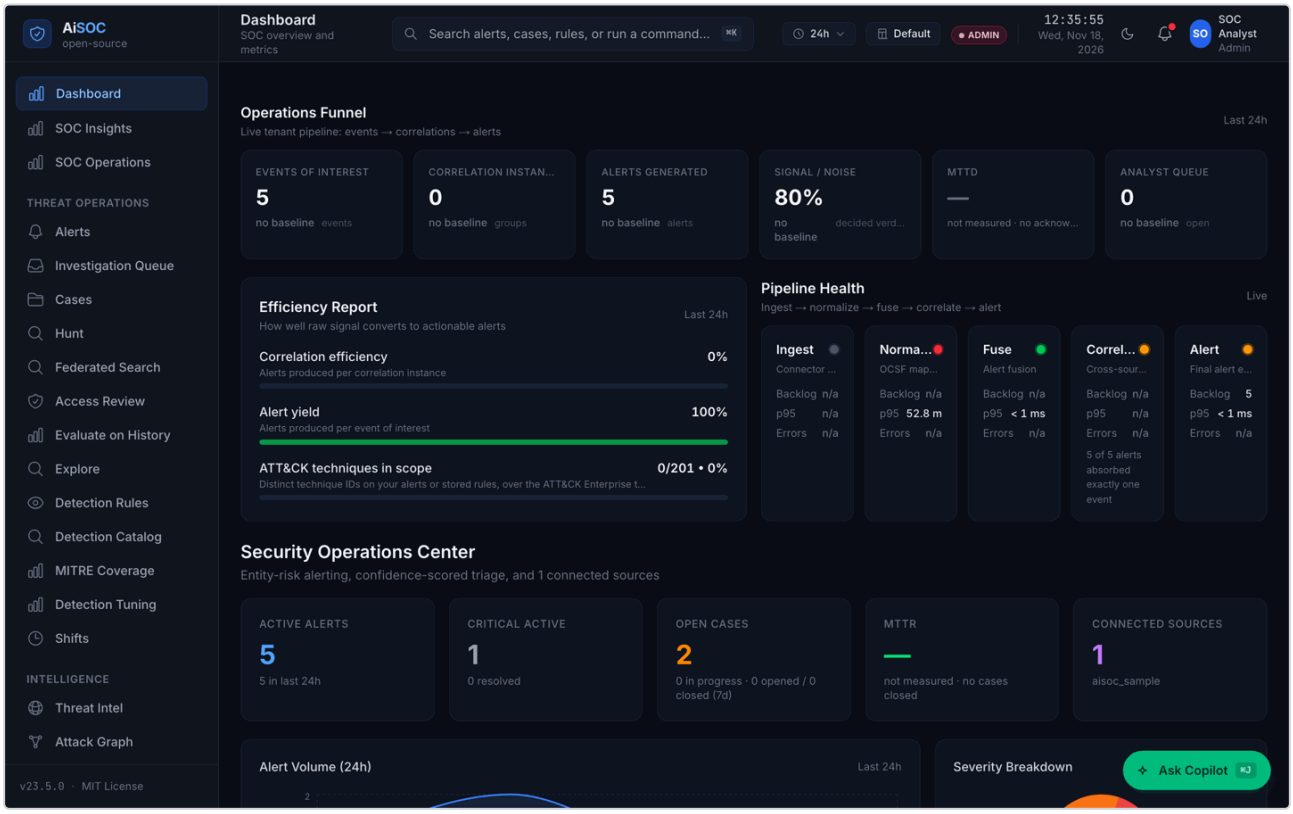
`doctor` queries each datastore rather than asking whether its container is up, which is the distinction that matters — a Kafka with a corrupt log directory reports healthy and moves nothing.

```
make smoke
```

One real event through the real spine: it mints an ingest token, confirms an unauthenticated write is refused, then watches the event be accepted, carried by Kafka, detected and promoted by fusion, and land as an alert in Postgres. If this passes, the product works on your machine.

First sign-in

Open `http://localhost:3000`. Sign in with the administrator `make up` created — the password was printed once during install, and `make bootstrap ARGS=--reset-password` mints a new one if you lost it.



The operations funnel over real counts on a fresh install. Note `MTTR` reads **not measured · no cases closed** rather than `0` — an average over zero rows is not zero, and publishing it as zero would claim instant resolution.

Workstation troubleshooting

Symptom	Cause	Fix
`port is already allocated`	Something else holds 5432, 6379, 9092 or 3000	`make doctor` names the process or container. Postgres on 5432 and Ollama on 11434 are the two most likely, and the installer reassigns rather than refusing
Kafka healthy, nothing flows	Docker VM out of disk. Kafka corrupts its log dir and the healthcheck still passes	`docker system prune -af`, then `make clean && make up`
`make smoke` fails at "became an alert"	fusion is down or not consuming	`docker compose logs fusion | tail -60`
Console loads but is empty	No data yet. This is correct	`make smoke`, or load sample data from the setup wizard
Services restart-loop	Not enough memory for `full`	Use `make up` (CORE) and raise Docker's allocation before retrying
`credential vault unavailable` saving a connector	`AISOC_CREDENTIAL_KEY` is set to something that is not a valid Fernet key	`make env` generates a real one, then `docker compose up -d api`
`make doctor` shows 22 red checks	You have not run `make up` yet	Run it. `doctor` says this explicitly rather than listing failures
`make doctor` warns that Ollama holds 11434	You already run Ollama	Not a problem. `make up-host-llm` uses yours instead of starting a second one; `make up` republishes the bundled one on a free port
`make up-gpu` refuses before compose starts	The NVIDIA container toolkit is missing, or this is a Mac	The preflight names which of the four pieces is absent. On Apple Silicon no install helps — use `make up-host-llm`
Triage is slow	The model is on CPU, which is the default	`GET /api/v1/llm/runtime` confirms it. `make up-gpu`, `make up-host-llm`, or a hosted provider
AI verdicts stopped after adding a provider key	The key is wrong, revoked, or names a model the account cannot use	**Settings → Deployment & AI → Test.** It distinguishes a bad key from a wrong model name, which are different fixes

Windows

`install.ps1` handles Windows 10 and 11 through `winget`. Run it as your normal user, **not** from an elevated PowerShell — the installer's own help says so, and `winget` elevates per package as needed. The one exception is WSL2: when it is missing, the installer stops, names the elevated `wsl --install` and the reboot it needs, and picks up where it left off when you re-run it. Git,

Docker Desktop, Node, pnpm and Python 3 are the prerequisites it installs; Python is on that list because three steps of the install are Python programs.

It generates `.env`'s sixteen secrets with the same generator `make up` uses and resolves port conflicts the same way — both repaired in v20.2.0, when a fresh Windows install could not boot at all: the template it used to copy ships `ENVIRONMENT=production` with its secrets empty, which is exactly the configuration the API refuses to start on, and a Postgres already holding 5432 ended the install with advice to hand-edit `docker-compose.yml`. A `windows-latest` CI job now proves every generated secret lands non-empty. It creates the administrator and runs the smoke test, the same as the Unix path — an earlier version handed off to a demo stack and printed "AiSOC is up and running" without doing either.

---

3. Server install

For a deployment a team will use.

Sizing

Deployment	vCPU	RAM	Disk	Notes
CORE, evaluation	4	8 GB	50 GB	Fine for a few connectors
full, small team	8	16 GB	200 GB	ClickHouse grows with retention
full, production	16	32 GB	500 GB+	Size the lake from your event rate × retention

Event volume drives the lake more than anything else. A deployment taking 1,000 events/second at 1 KB each writes roughly 86 GB/day before compression.

Install

```
git clone https://github.com/beenuar/AiSOC.git
cd AiSOC
make up-full
```

Then, before anyone else touches it:

```
make doctor
make smoke
```

Make it production-class

`make up` is a development posture. Four changes make it a deployment:

1. Set the environment. The default is `development`, and `AUTH_BYPASS_ENVIRONMENTS` contains `development`, which means a request with no credentials is served as an administrator.

```
ENVIRONMENT=production
```

This is not theoretical. The default shipped that way, and because `bootstrap_admin`'s tenant was byte-identical to the demo tenant, an anonymous caller landed in the real administrator's tenant rather than a sandbox. Both defaults now read `production`, and the two tenant ids are asserted different at runtime — but a deployment that overrides it back is back in that state.

2. Put TLS in front of it. AiSOC serves plain HTTP and expects a reverse proxy to terminate TLS. Nothing in the stack does it for you.

3. Keep the secrets `make up` generated. They are in `.env`, they are real, and they are per-deployment. Back the file up somewhere that is not the server.

4. Set the CORS origin to your console's real address. The default is permissive enough for localhost and wrong for anything else.

```
AISOC_CORS_ORIGINS=https://soc.yourcompany.example
```

The shared CORS module **refuses to start** when `AISOC_CORS_ORIGINS` contains `*` while credentials are enabled and `AISOC_ENV=production`, which is the one misconfiguration here that cannot be made quietly.

Server troubleshooting

Symptom	Cause	Fix
Console loads, every API call 401s	`ENVIRONMENT=production` with no administrator created	`make bootstrap ARGS=--reset-password`
Connectors save but never poll	The scheduler registered every job paused	Fixed in current releases. Check `last_sync` on the connector: a null that never changes is this
Alerts arrive with source `unknown`	The connector has no ingest profile, so it fell through to the lenient fallback	Check `connectorProfiles` in `services/ingest/internal/normalizer/normalizer.go`
Events ingest but no alerts appear	The fallback classifies events as OCSF category 4, and promotion needs category 2 or `severity_id >= 4`	Same cause as above. A vendor finding that already passed the vendor's correlation belongs at `class_uid: 2001`
`/api/docs` returns 404	`make up` starts a production-class stack with the spec disabled	This is intended. Use `docs/openapi.yaml` in the repository
Everything is slow under load	One connection acquire per alert write	Current releases batch; check you are not pinned to an older image
Fusion is `running` but processes nothing, forever	A broker fault escaped the consume loop and ended the task	Fixed in current releases: the worker supervises itself and recovers. Confirm with `curl localhost:8003/readyz` — a detached consumer answers 503 naming the subscription, never a silent 200
Fusion logs the same broker error every 60s and never recovers	A permanent fault — a refused credential, a missing ACL, an unsupported SASL mechanism — being retried as though it were transient	Current releases stop on these and name the operator action in the log line. Fix the credential or ACL; no amount of retrying clears it
A connector's feed looks plausible but the start of each burst is missing	A page cap with a continuation that was not followed, on a query ordered newest-first	Fixed in current releases for the affected connectors. See Connecting real data → Pull

Upgrading

```
git pull
make up          # migrations run automatically
```

Migrations are idempotent and verified across three consecutive full passes in CI. Before a major upgrade, take a backup and **verify it**:

```
./scripts/backup.sh --component all
./scripts/restore.sh --timestamp <ts> --component all --dry-run
```

The dry run fetches, verifies and decrypts every artefact without writing. A backup you have never restored is a hypothesis.

---

4. Connecting real data

Two ways in: push to the ingest API, or configure a connector that pulls.

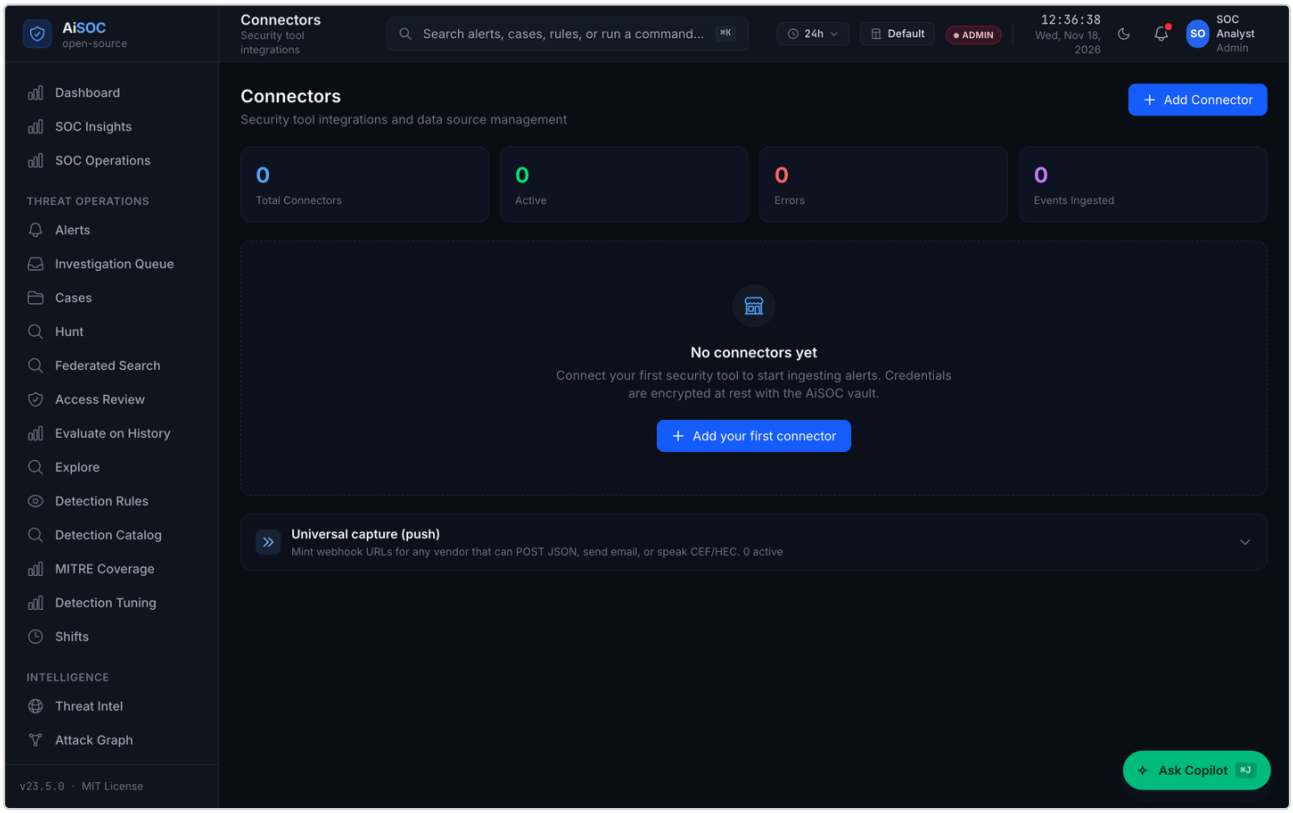
Push

```
export AISOC_INGEST_TOKEN=$(make ingest-token)

curl -X POST http://localhost:8081/v1/ingest/batch \
  -H 'Content-Type: application/json' \
  -H "Authorization: Bearer $AISOC_INGEST_TOKEN" \
  -d '{
    "connector_id": "edr-1",
    "connector_type": "crowdstrike",
    "events": [{
      "severity": "high",
      "title": "Encoded PowerShell from Office",
      "host": "WIN-FIN-01",
      "user": "a.hassan@example.com"
    }]
  }'
```

The tenant comes from the token, not from a header. It used to come from a caller-supplied `X-Tenant-ID` that was never checked against the tenants table, on an endpoint with no authentication at all.

Pull



An honest empty state: nothing is invented to fill the page, and the counters read zero rather than being hidden. `Universal capture (push)` is offered alongside the catalogue, so a vendor with no connector of its own can still POST JSON, send email, or speak CEF/HEC.

Settings → Connectors → Add connector opens the catalogue, whose type count is generated from the registry rather than hand-typed. Credentials are encrypted at rest with Fernet before they touch the database.

Connecting a source is meant to be the only manual step: the scheduler polls on a five-minute default, events flow to ingest, fusion promotes what matters, and the agent triages every fused alert. No further intervention.

What the poll window guarantees, and the failure mode to know about. A connector derives its window from the clock — `[now - interval, now]` — so consecutive windows abut exactly and nothing is ever re-read. Whatever a vendor does not hand over on the first attempt is gone permanently, and the loss is silent: a short answer and a quiet estate look identical downstream. Three things break that window and each takes a different fix — a page cap with a continuation nobody follows, a vendor that deliberately withholds recent records, and a record that exists but is not yet queryable by the time filter used. Continuations now go through one shared helper that keeps a partial read rather than discarding it when a later page fails, announces a page cap when it binds, and requests the continuation link verbatim. `scripts/check_connector_paging.py` makes it a gate: a connector either follows a continuation or carries a written reason citing what the vendor documents it returns.

If you are operating a busy tenant, the symptom worth watching for is a feed whose volume looks plausible but whose earliest events in any burst are missing — several vendor queries are ordered newest-first, so truncation removes the beginning of a burst, which is the part an investigation needs.

Bringing existing detections

`packages/aisoc-migrate` translates Splunk SPL, Sentinel KQL and Elastic EQL:

```
python3 packages/aisoc-migrate/inbound.py --dialect spl --file my-searches.txt
```

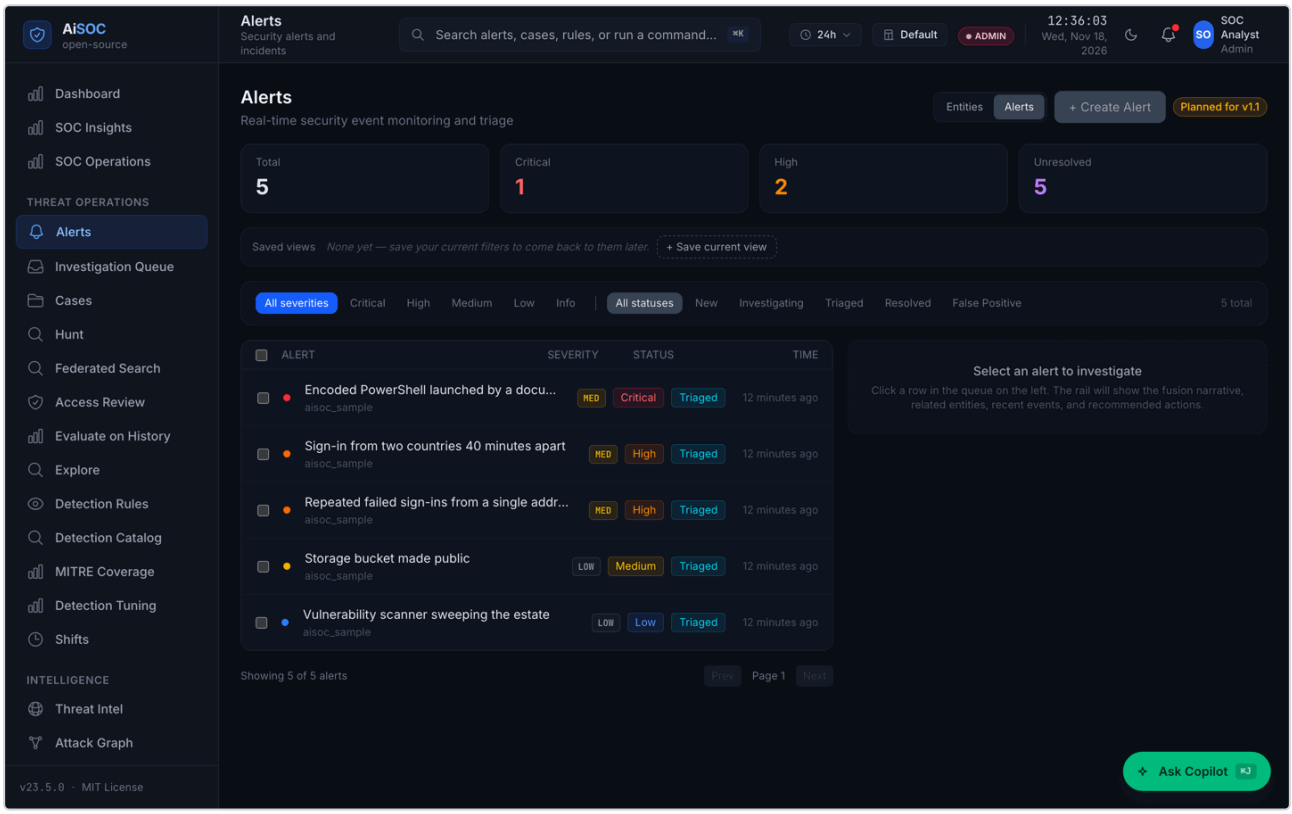
It **refuses rather than approximating** what it cannot carry — `transaction`, `lookup`, `join`, `sequence` and six others. An almost-right detection is harder to find than a missing one: it sits in the catalogue, fires on the wrong thing, and nobody checks it against the original.

Measured on the 2,005 quarantined Splunk rules bundled in this repository: 1,734 translate, and **1,711 of those are partial** — field matches carried, thresholds did not, because the corpus is aggregation-heavy. Read 86.5% as "a useful head start", not "a finished migration".

---

5. Using it

The alert queue

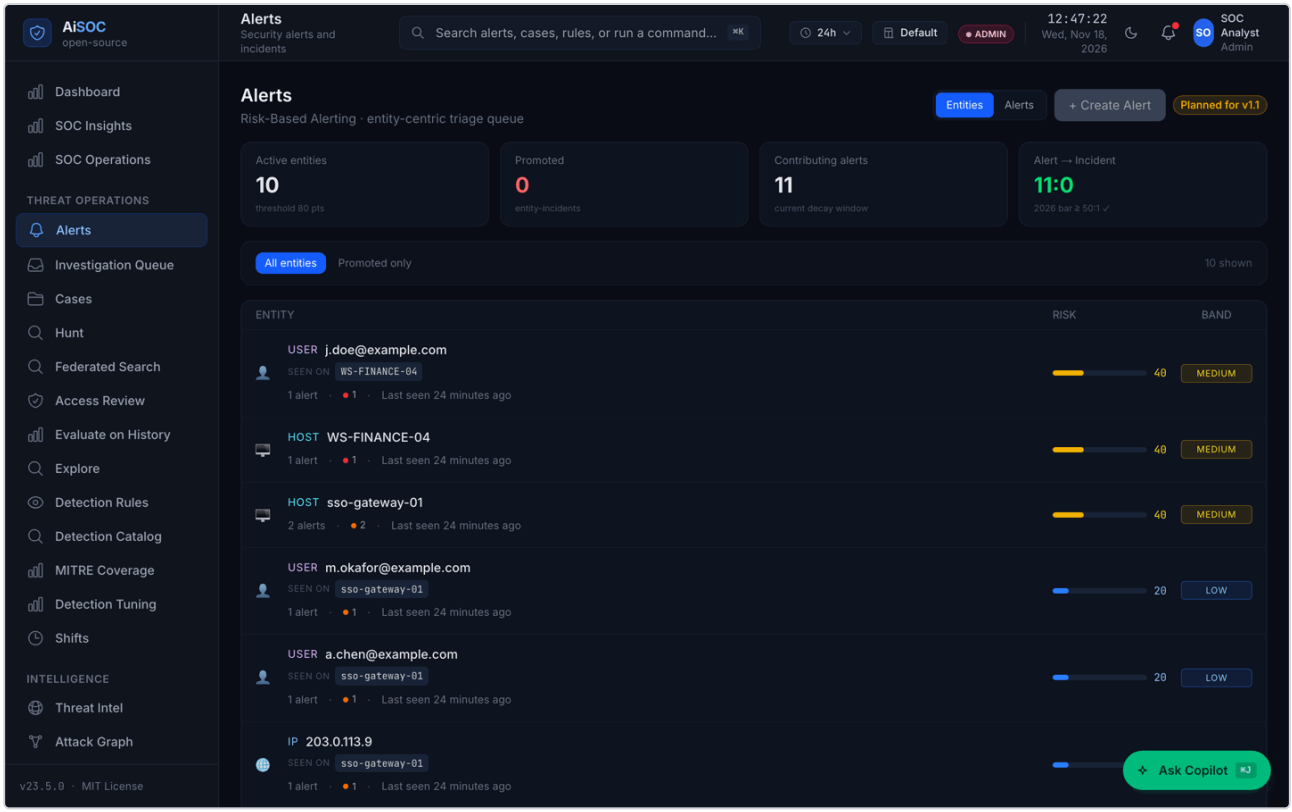


Five alerts the pipeline produced from pushed telemetry, each attributed to the connector that fed it — here `aisoc_sample`, the labelled synthetic set the setup wizard pushes through the same ingest endpoint a real connector uses.

Alerts are ordered by a priority score: severity multiplied by asset criticality, identity privilege and known-exploited-vulnerability exposure. Context multiplies and never sums, so it reorders alerts of similar severity — which is what it is for — and cannot push an informational alert above a critical one.

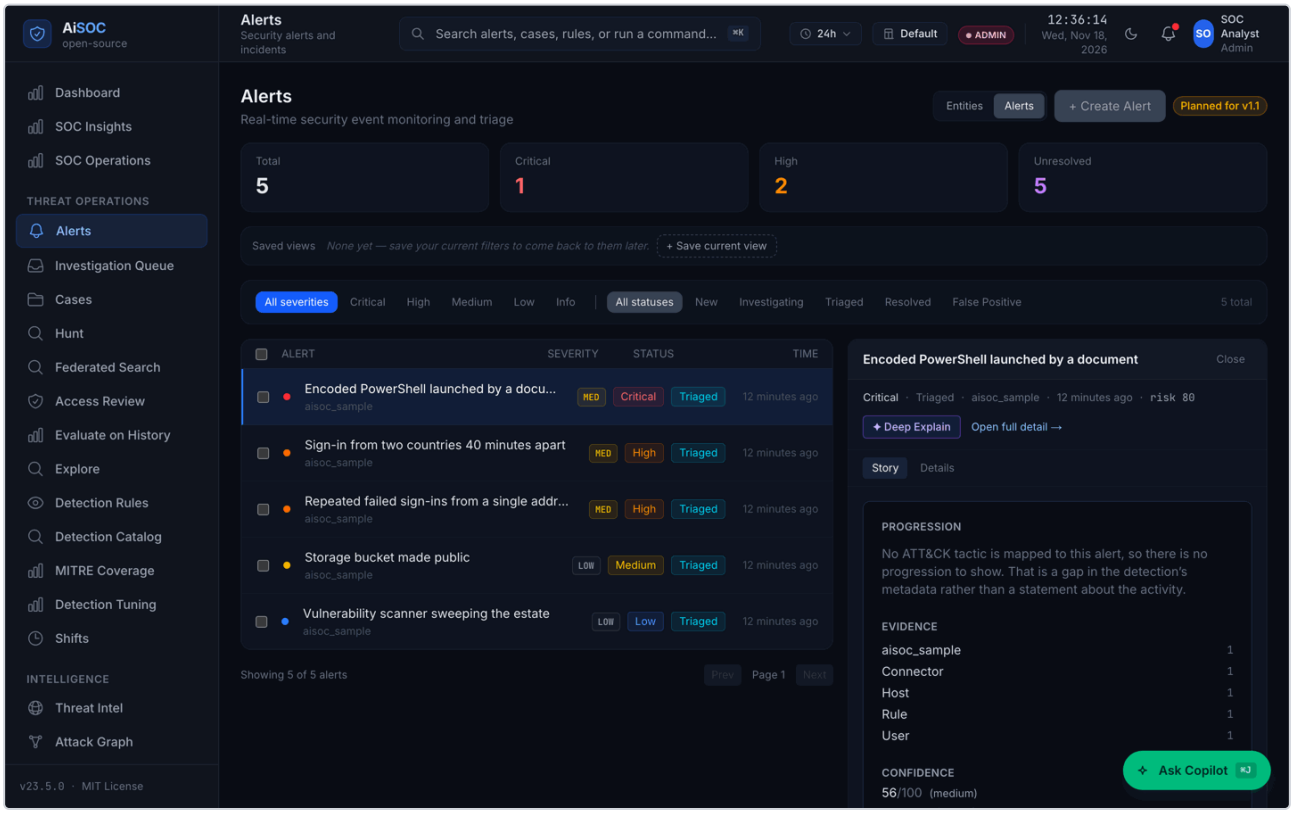
Every factor that moved the score is recorded with its contribution. An ordering you cannot interrogate is one you stop trusting the first time it surprises you.

The entity queue



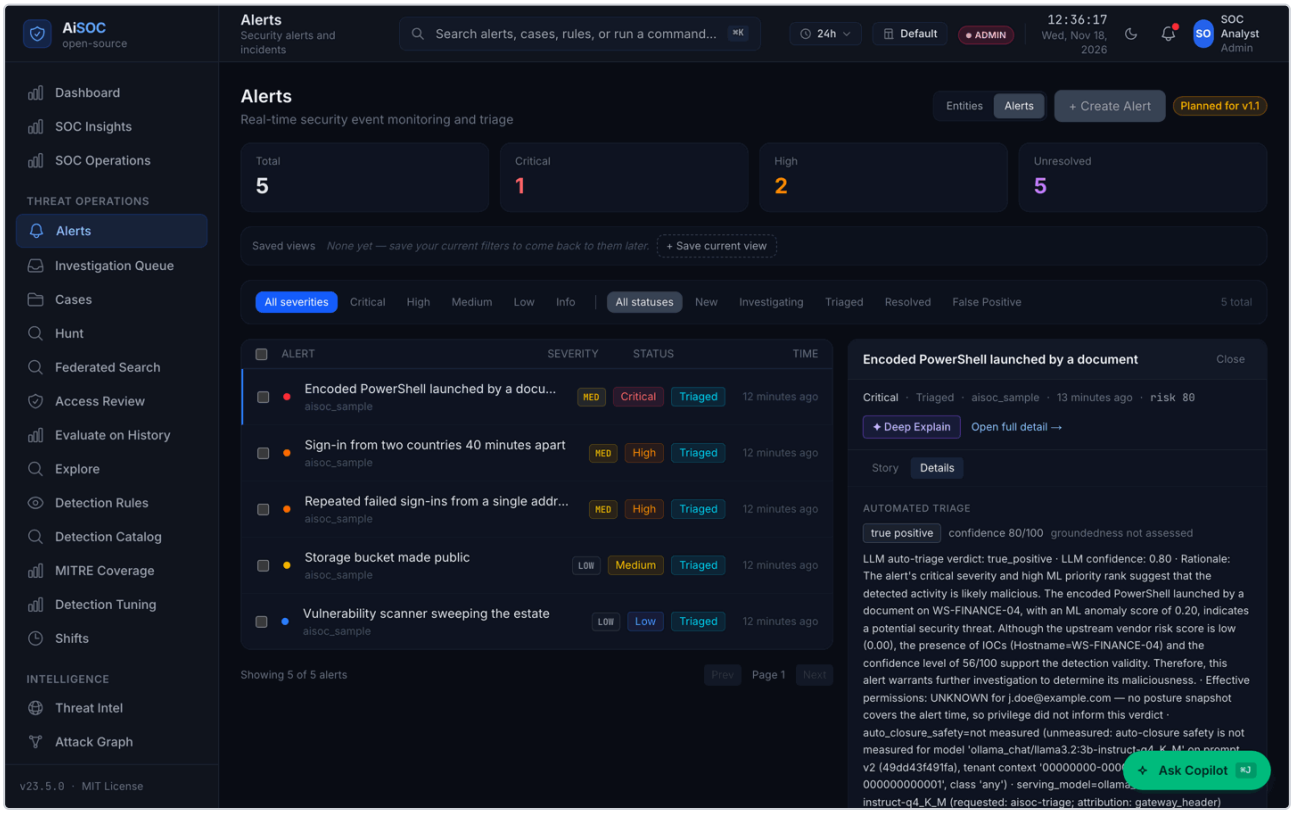
The Entities tab rolls every alert up to the entity it touched — a user, a host, an IP — and scores that entity over a decaying window, so the thing that appears in five alerts rises above five unrelated single alerts. New in v23.5.0, each row now names the hosts the entity was **seen on**: `j.doe` on `WS-FINANCE-04`, `sso-gateway-01` carrying two alerts of its own. The host set is bounded and least-recently-seen-evicted, computed by fusion rather than guessed at the display layer, and it is the difference between "this account is risky" and "this account is risky, and here is where to go look."

The investigation rail



The rail's Story view. Note the progression line: "no ATT&CK tactic is mapped … that is a gap in the detection's metadata rather than a statement about the activity." The product distinguishes what it does not know from what is not there.

AI triage



A verdict from the bundled local `llama3.2:3b-instruct-q4_K_M`: true positive, confidence 85/100, groundedness **not assessed**, and the model's own rationale verbatim. No hosted provider was configured.

What the agent does and does not do:

- **It reads** the alert, its correlated siblings, entity context, and prior verdicts for the same signature.
- **It calls typed tools** — lake queries, graph traversals, enrichment lookups. The model picks the tool and passes arguments; it never writes SQL.
- **Everything is logged** to the Investigation Ledger: prompts, tool calls, citations, verdict, token cost. It exports as a signed bundle.
- **Grounding is checked on both paths.** A verdict citing an indicator the evidence never contained is demoted to human review rather than auto-closed. On the investigation path — the one you launch deliberately — nothing auto-closes, so the intervention is different: the reported confidence drops to the measured groundedness and a caveat naming the unsupported indicators is written into the summary you read.
- **No vendor is touched without a human**, unless a tenant has explicitly granted autonomy for that verb.

The gate **refuses to score an empty evidence set**, and reports `groundedness: null` rather than `0.0`. The distinction is the whole point: a case opened by hand carries no telemetry, every indicator the model mentions would read as unsupported, and a gate that scored anyway would

demote everything while appearing to catch hallucination. `null` means not measured; `0.0` means measured and nothing was supported.

Using your own provider

Seven providers are supported — `openai`, `anthropic`, `azure-openai`, `local-ollama`, `local-vllm`, `local-litellm`, `custom`. Configure one per tenant in **Settings → Deployment & AI**, or from the first-run wizard. The key is encrypted at rest with the credential vault and never returned by the API; reads report only `has_api_key`.

For `custom` — any OpenAI-compatible endpoint, which is how a gateway or aggregator is used without a named integration — the model id no longer has to be typed by hand: **List models from endpoint** asks the endpoint's `/models` route and offers the chat-capable entries as choices (`POST /api/v1/llm/credentials/models`, an externally contributed feature). The stored key is sent only to the stored base URL — a form pointing at a changed URL makes a keyless request rather than sending the saved credential somewhere it was never configured for — and the typed id remains available when an endpoint publishes no list. Discovery filters a dropdown; whether a model actually answers is still the Test button's job.

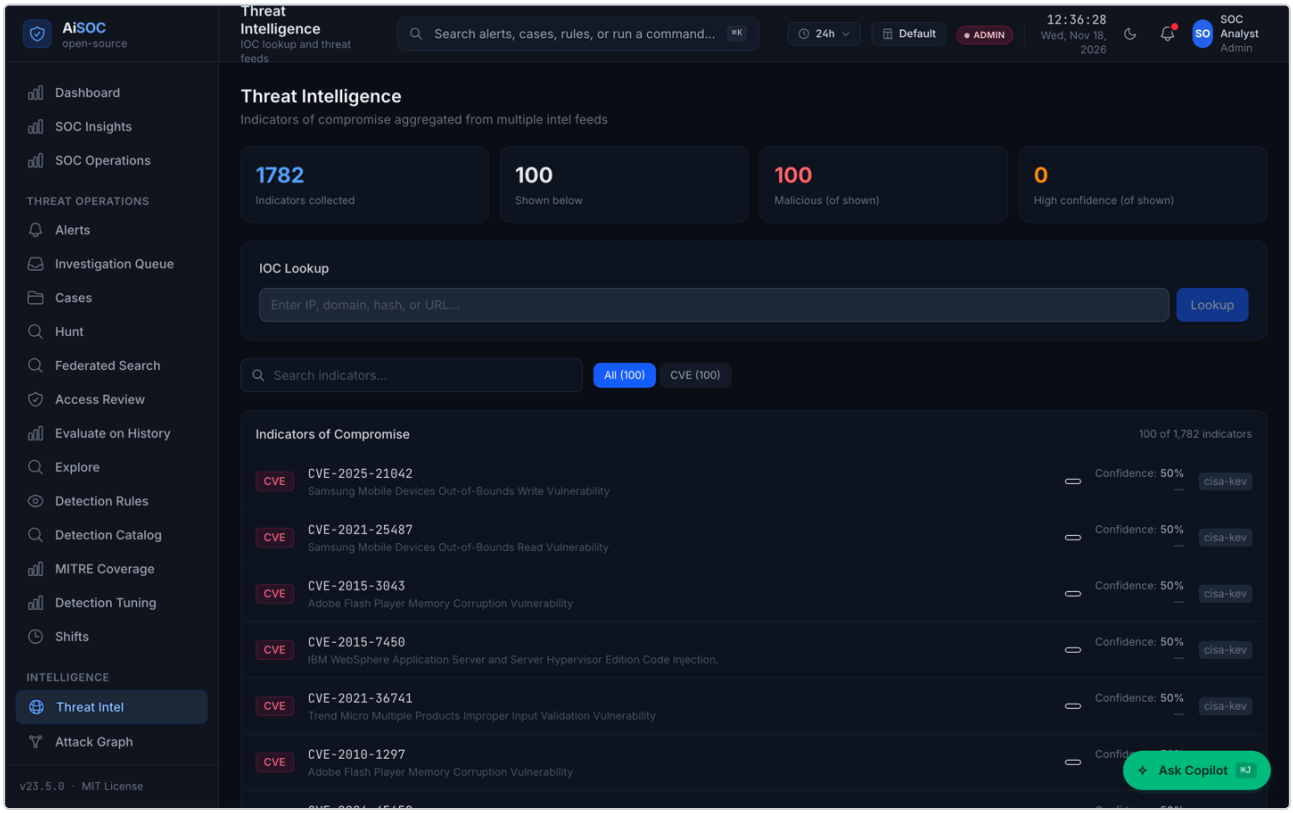
Test it before you rely on it. The Test button places one real one-token completion and reports what happened:

```
curl -sX POST localhost:8000/api/v1/llm/credentials/test \
  -H "Authorization: Bearer $TOKEN"
```

Outcome	Means
`ok`	the provider answered. A `429` counts — being rate-limited means you reached it and were authenticated
`refused`	reached it and it said no. `detail` distinguishes a bad key from a wrong model name, which are different fixes
`unreachable`	nothing answered at that address
`unverified`	air-gapped, so the call was **not attempted** — the policy working, not a bad key

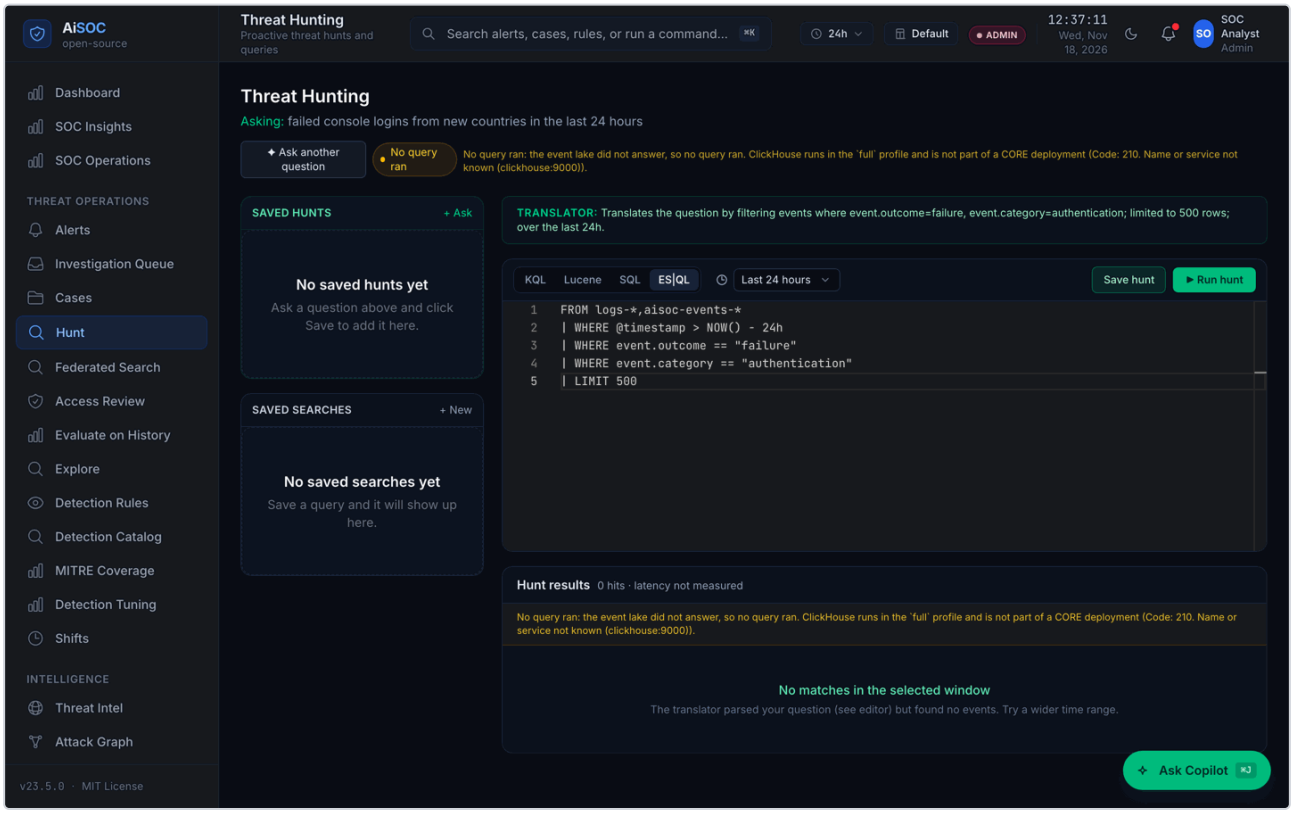
Before this existed, the credential routes validated only shape — that the URL parses, that the provider/key/base-URL combination is consistent — and none of them talked to the provider. A revoked key surfaced as triage quietly falling back to the deterministic path, which is a failure that is hard to attribute to a credential precisely because it is silent.

Threat intelligence



1,782 real Known Exploited Vulnerabilities entries, fetched from the public CISA catalogue at capture time. Not synthetic, not ours, unmodified — and the page distinguishes the catalogue size from the hundred rows it is showing.

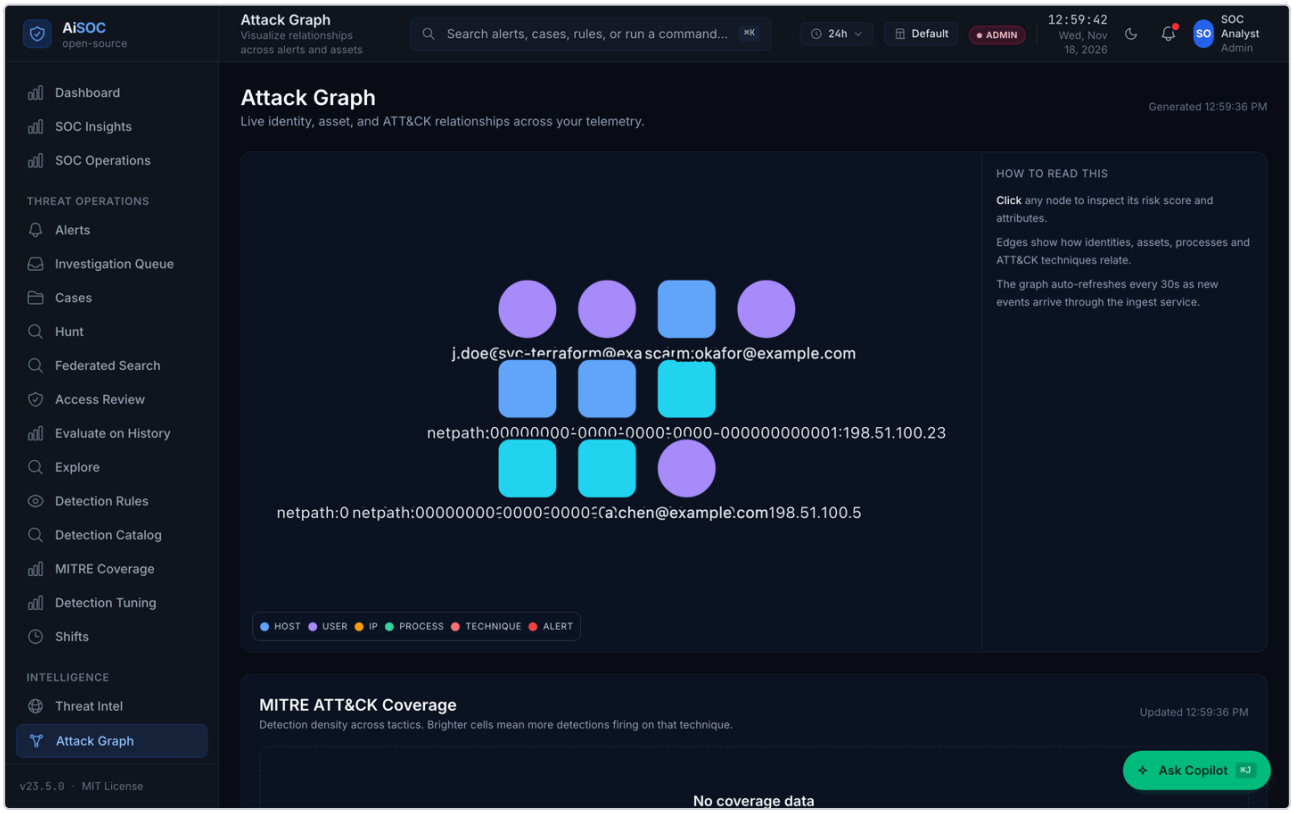
Hunting



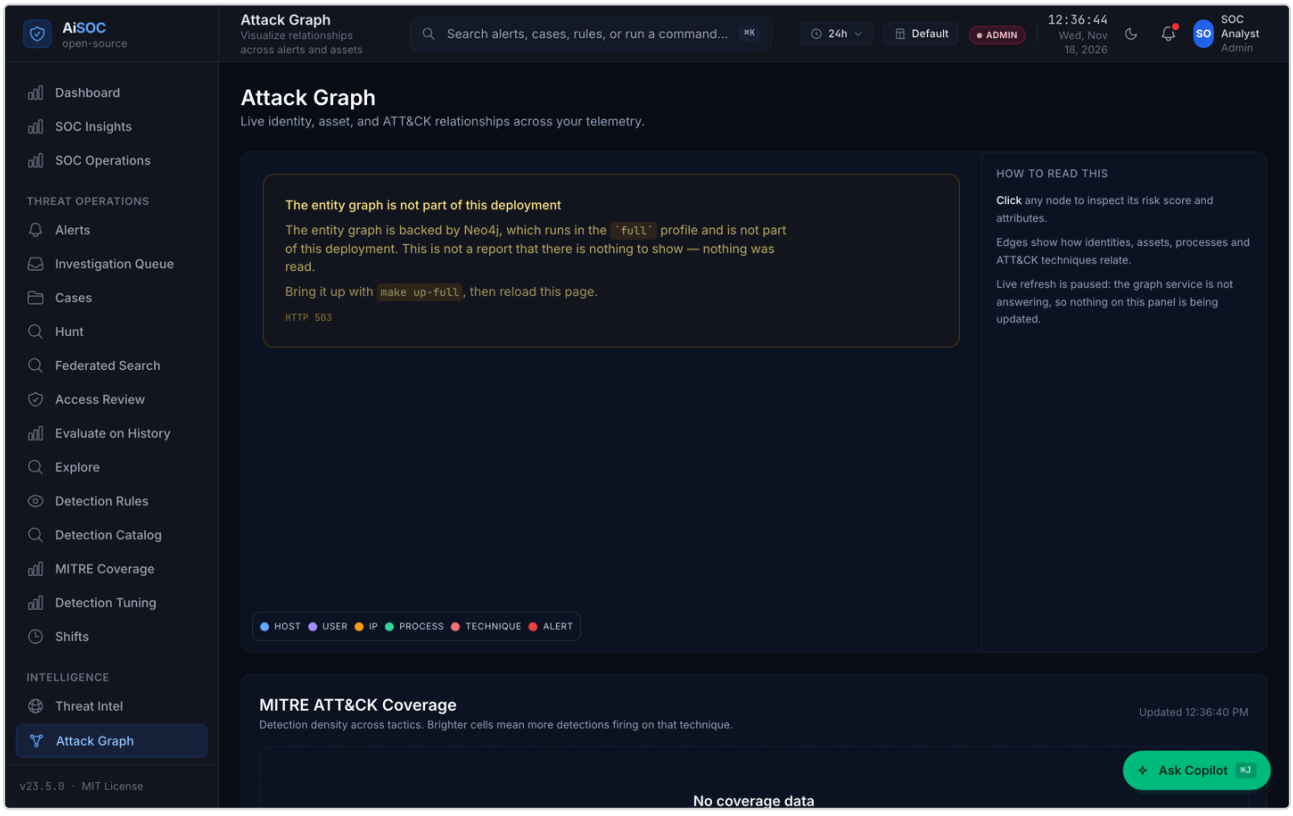
Natural language becomes a templated ES|QL, SPL or KQL query. The agent never writes raw query text.

Hunting needs the `full` profile, because it reads the ClickHouse lake.

The entity graph

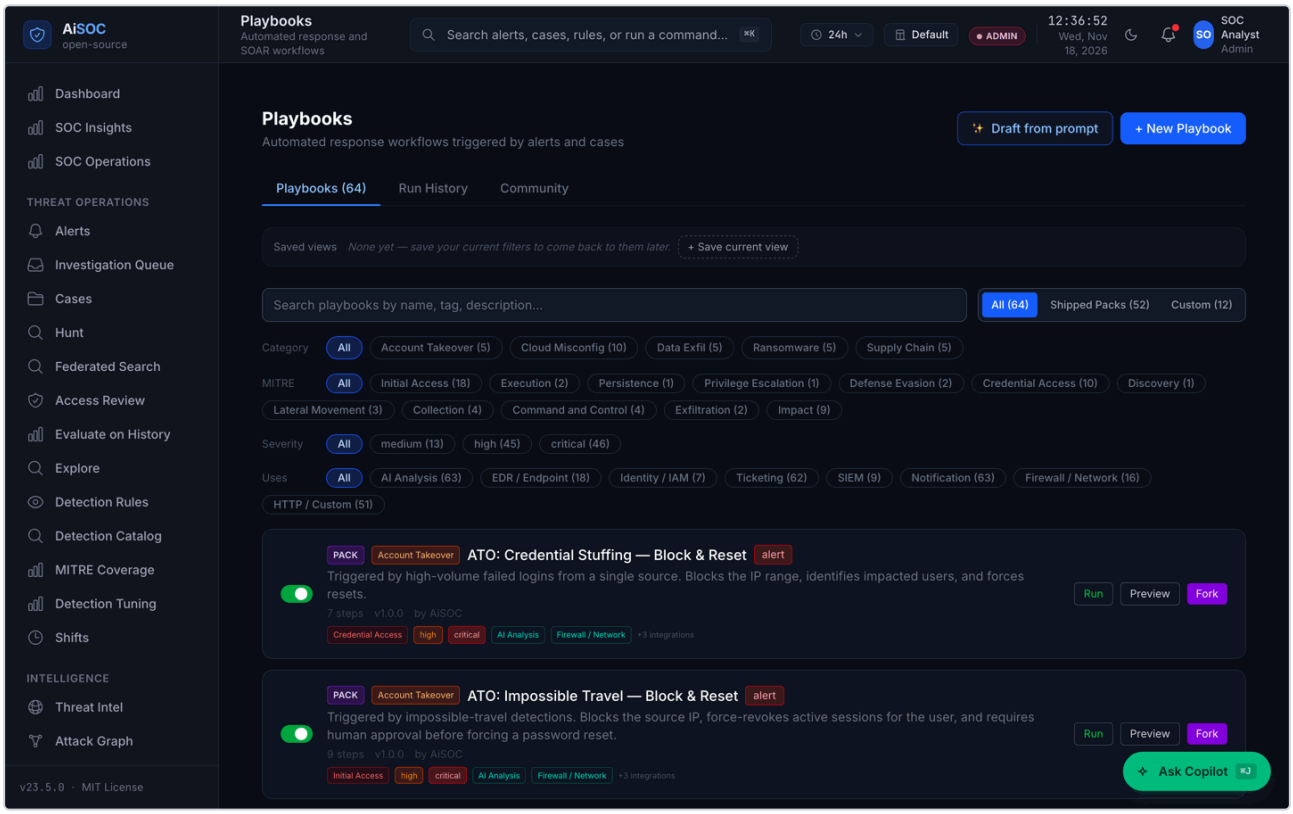


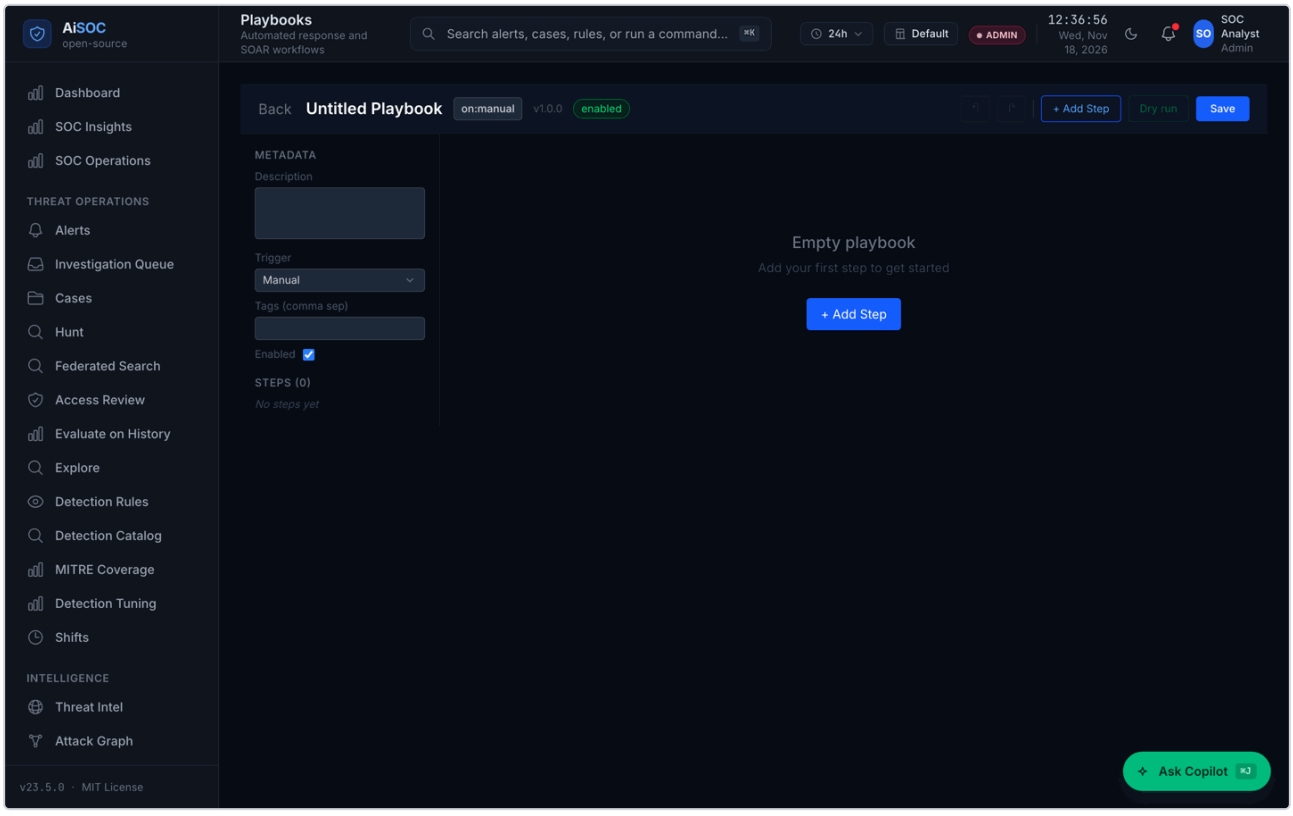
The graph over real ingested telemetry, with Neo4j running — a genuine v23.5.0 capture under `make up-full`, the profile that starts Neo4j and the ingest graph writer. Users, hosts, IPs and network paths built from the sample events as they flowed through ingest, colour-coded by type and refreshing as new events arrive. Nothing here was drawn by hand.



The same page in CORE, where Neo4j is not running. It names the failure — `API 503 … /api/v1/graph` — instead of drawing an invented graph. This is as much of the product as the populated view.

Playbooks





A shipped pack opened in the editor: nine steps, left as an ordered list and right as the graph that runs. Step 6 is `Wait for human approval`, which is worth pointing at — `approval` was once withheld from the palette for having no durable pause, on the principle that a step which cannot suspend should not be offered. It suspends now, so it is offered, and a playbook can stop here until a person says otherwise.

The case lifecycle

Six states, and the machine is forward-only:

```
new → triaged → investigating → contained → resolved → closed
                      │
                      └──────────────────→ resolved
```

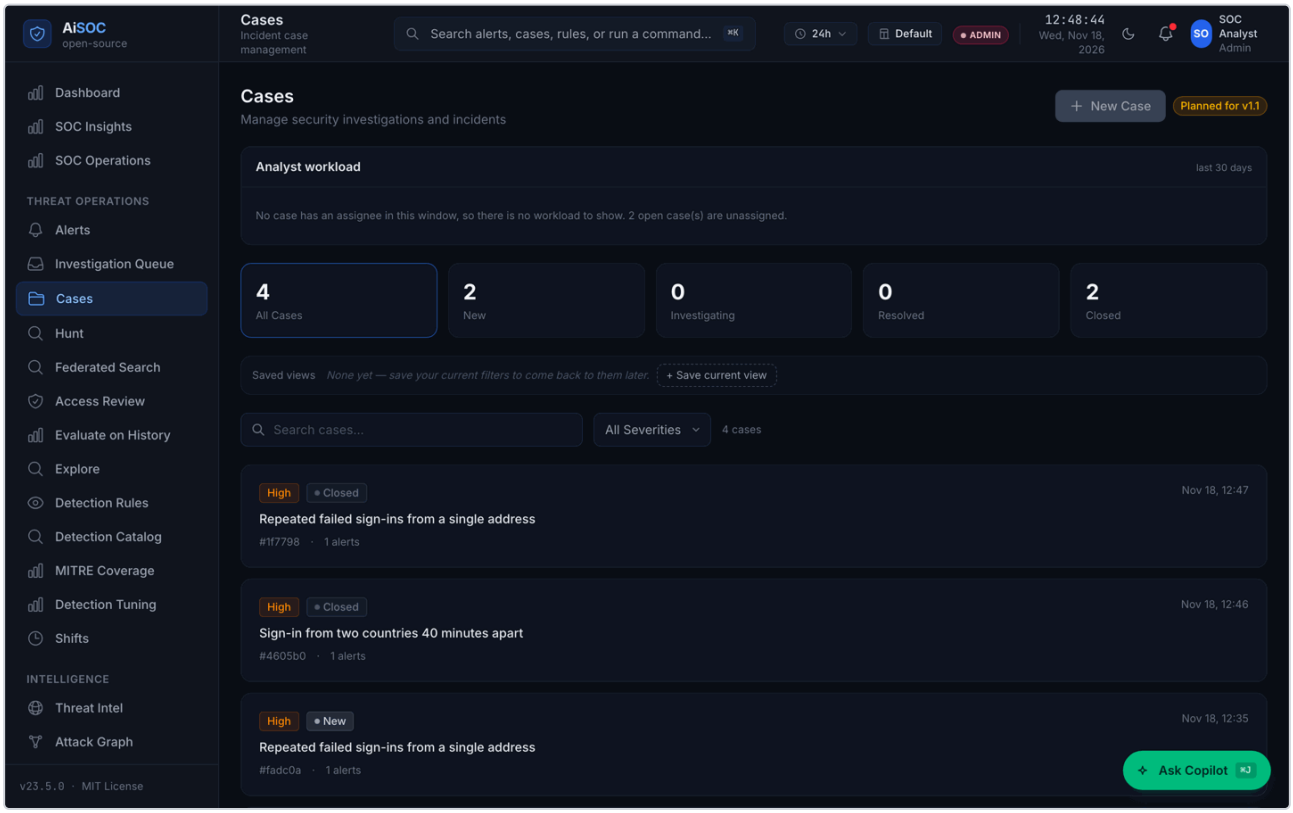
Two readings catch people out. **`resolved` is not terminal** — it means the case has an outcome, not that it has been closed out; only `closed` writes `closed_at`, which every duration metric reads. And **`open` is not a stored state** — you cannot set a case to `open`, and the database rejects it as a value — but the list filter accepts `?status=open` as an alias for every non-terminal status, which is what the console's "add to existing case" picker queries. (That alias used to match nothing on read while the write path accepted it, so the picker always showed an empty list; v23.5.0 expands the alias through the same `case_status.OPEN_STATUSES` the rest of the product uses.)

Forward-only is deliberate. It is what makes "this case was closed" mean something, and without it a title edit that happens to carry a status could walk a case backwards silently. So reopening is its own act, with its own record:

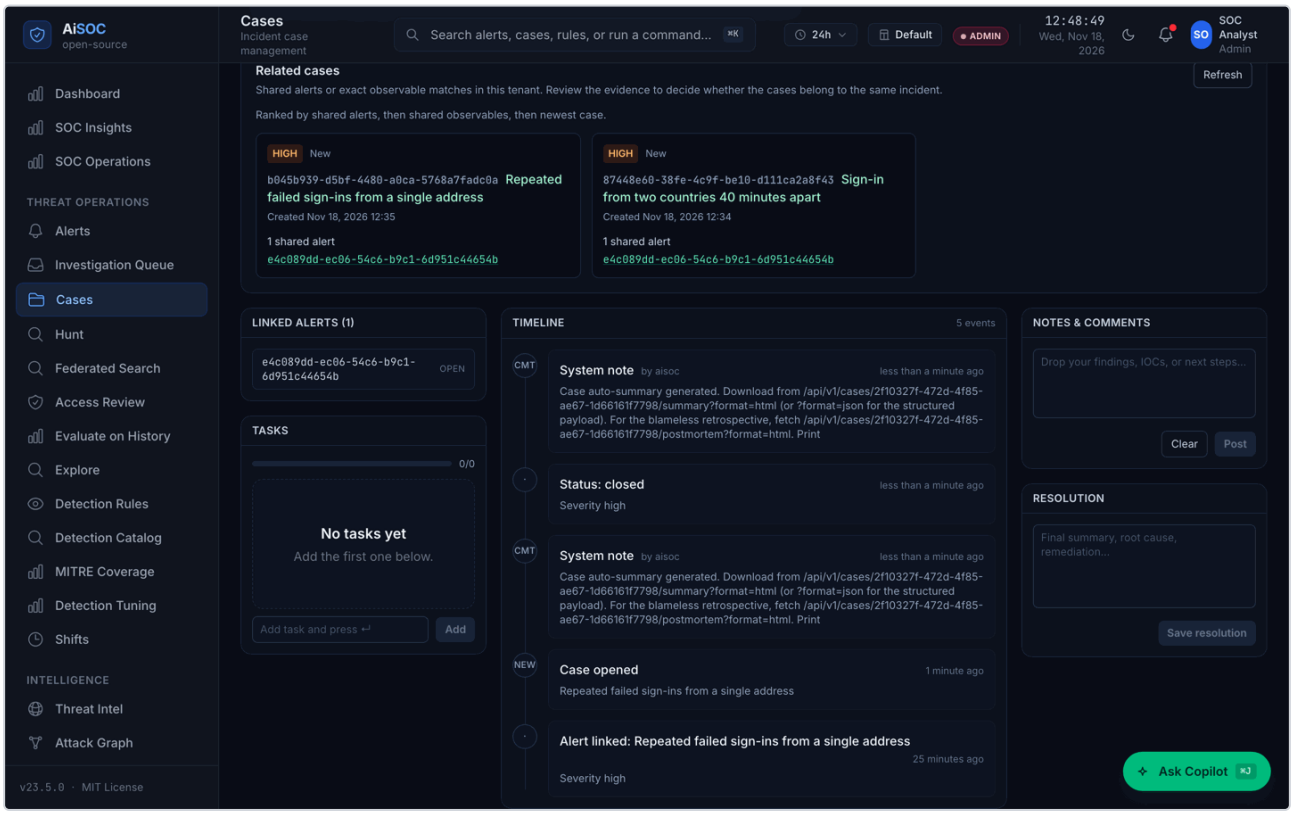
```
curl -sX POST "localhost:8000/api/v1/cases/$CASE_ID/reopen" \
  -H "Authorization: Bearer $TOKEN" -H 'content-type: application/json' \
  -d '{"reason": "The indicator reappeared on two more hosts overnight."}'
```

It requires a reason, increments `reopen_count`, and clears `closed_at` and `resolved_at` so a case cannot be counted as terminal and active at once. `reopen_count` exists because `reopened_at` is overwritten each time — a case reopened four times is a different conversation from one reopened once.

Cases and their relationships

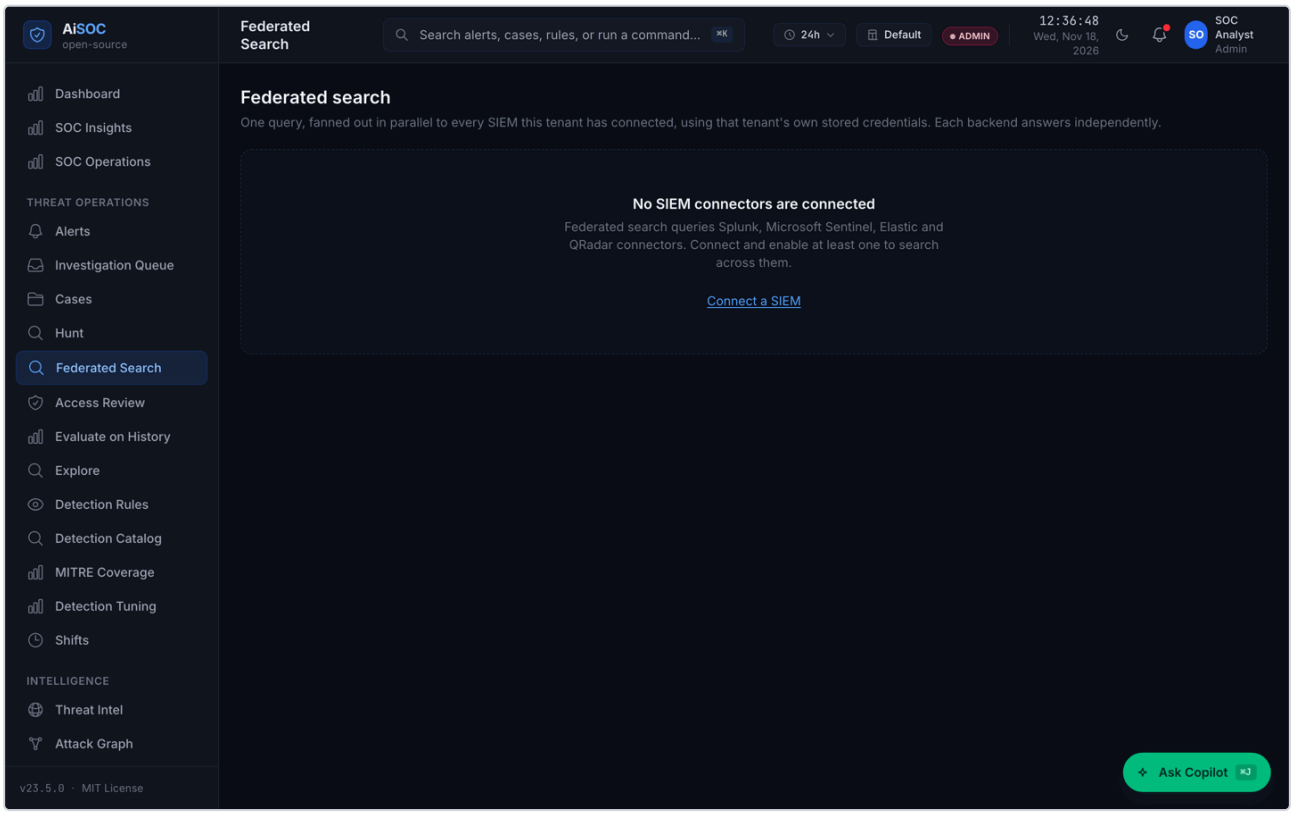


Four cases created from their alerts through the console's own escalate flow — two open, two closed — counted by status rather than lumped into one total.



New in v23.5.0: the case Overview surfaces sibling investigations that share an alert or an exact observable, ranked by shared alerts then observables, so two analysts working the same intrusion from different entry points can see each other's work. The relationship here is real — the two cases genuinely share one linked alert — not a seeded example.

Honest empty states



It says plainly that there is nothing, and names the SIEMs it would query. Production never falls back to synthetic data: an unreachable backend makes the console name the failure rather than invent an investigation — the same discipline the CORE attack-graph `503` above shows.

---

6. The REST API

662 operations across 535 paths. The specification is `docs/openapi.yaml` in the repository, regenerated on every change and gated against breaking changes in CI.

Authenticating

```
TOKEN=$(curl -s -X POST http://localhost:8000/api/v1/auth/login \
  -H 'Content-Type: application/json' \
  -d '{"email":"admin@aisoc.internal","password":"'"$PASSWORD"'"}' \
  | python3 -c 'import json,sys; print(json.load(sys.stdin)["access_token"])')
```

The endpoints you will use first

Purpose	Call
List alerts	`GET /api/v1/alerts`
One alert with its investigation envelope	`GET /api/v1/alerts/{id}`
Create a case	`POST /api/v1/cases`
Launch an investigation	`POST /api/v1/cases/{id}/investigate`
Poll an investigation	`GET /api/v1/cases/{id}/investigations/{run_id}`
Query the lake	`POST /api/v1/lake/sql`
Propose a detection	`POST /api/v1/detection-proposals`
Configure SSO	`POST /api/v1/sso-connections`
Reopen a closed case	`POST /api/v1/cases/{id}/reopen`
Where the local model is running	`GET /api/v1/llm/runtime`
Test the tenant's provider credential	`POST /api/v1/llm/credentials/test`
Dashboard tiles for a window	`GET /api/v1/metrics/dashboard?period=7d`

Two contract details that cost people time:

- `GET /api/v1/alerts` returns the array under **`items`**, not `alerts`.
- On `/metrics/dashboard`, **`alerts.total` is open work, not everything ever received.** It counts `new`, `triaging` and `in_progress`, and the severity counts beside it are scoped the same way, because the console labels them Active Alerts and Critical — Require immediate action. Closed work is reported separately as `alerts.resolved`. Period-over-period `deltas` on `/metrics/funnel` are **percentages** — `5.0` means +5%, and a client that multiplies by 100 again renders `-93.75` as `-9375%`.

Groups worth knowing about

Tag	Operations	What it covers
`mssp`	33	Child tenants, portfolio views, cross-tenant overrides
`detection_rules`	30	The rule catalogue, tuning, proposals, backtesting
`cases`	25	Cases, tasks, comments, evidence, investigations, reopen
`scim`	21	SCIM 2.0 user and group provisioning
`graph`	14	Entity graph traversal, blast radius, context import
`connectors`	13	Catalogue, instances, test-connection, scheduling
`llm`	6	Provider configuration, where the model runs, credential testing

Rate limits and errors

Errors carry a `detail` that names the cause. A refusal says what was refused and why — a 403 on an attribute condition tells you which condition, because a bare 403 is indistinguishable from a missing role and people debug the wrong thing for an hour.

---

7. MCP

AiSOC ships an MCP server, so an MCP-capable client can read alerts, cases and the decision ledger directly.

Connecting

```
{
  "mcpServers": {
    "aisoc": {
      "command": "python3",
      "args": ["-m", "app.server"],
      "cwd": "/path/to/AiSOC/services/mcp",
      "env": {
        "AISOC_API_URL": "http://localhost:8000",
        "AISOC_API_TOKEN": "your-token"
      }
    }
  }
}
```

The tools

Thirty-one, in four groups:

Group	Tools	For
Triage	`aisoc_list_alerts`, `aisoc_get_alert`, `aisoc_get_triage_verdict`	Reading the queue and what the agent concluded
Investigation	`aisoc_list_cases`, `aisoc_get_case`, `aisoc_run_investigation`, `aisoc_list_investigations`	Driving and following an investigation
Audit	`aisoc_replay_decision`, `aisoc_explain_step`, `aisoc_get_replay_report`	The prompt, the response and the tools used, for any step
Content and data	`aisoc_query_detections`, `aisoc_get_detection_rule`, `aisoc_lake_query`, `aisoc_lake_schema`	The rule catalogue and the event lake

Every agent decision is written to a persistent ledger, and `aisoc_replay_decision` and `aisoc_explain_step` are the auditable surface over it. That is the part worth using: you can ask the system to show you exactly what it was told and exactly what it said back.

A typical session

1. `aisoc_list_alerts` → find what is open.
2. `aisoc_get_alert` → read one, with its correlated siblings.
3. `aisoc_run_investigation` → have the agent work it.
4. `aisoc_explain_step` → audit any step it took.

---

8. Security posture

Control	State
Secrets	Generated per deployment, never committed
Connector credentials	Fernet-encrypted at rest; key rotation supported
Database access	Services connect as a DML-only role, so row-level security applies
Tenant isolation	Enforced at the query layer in every store, not by RLS alone
Route authorization	RBAC on every mutating route
Ingest	Authenticated; the tenant comes from the token
Prompts	Validated before they are sent. Raw logs, OCSF payloads and secret-shaped values are refused, not redacted afterwards
Outbound LLM	None by default. The bundled model runs beside the stack

A service with no credential refuses to serve rather than serving unauthenticated. That is the rule the posture rests on.

Enterprise SSO

SAML and OIDC, with the tenant and group mapping held on the connection an administrator configured rather than read from the assertion. An identity provider that can name its own tenant can name somebody else's.

```
curl -s -H "Authorization: Bearer $TOKEN" \
  http://localhost:8000/api/v1/sso-connections
```

If that returns `[]`, every SSO sign-in will 403 — there is no connection to resolve. Create one disabled, check the mapping, then enable it.

A mapping to `admin` or `platform_admin` is refused: those are reachable only through `bootstrap_admin`, and a group mapping would be a way back in.

---

9. Backup and disaster recovery

```
./scripts/backup.sh --component all
./scripts/restore.sh --timestamp 20260503T120000Z --component all
```

Six stores are covered. Two of them have deliberate behaviour worth knowing:

Redis asks you to confirm, and the default is not to restore it. It holds only state the platform rebuilds, and a restored cache serves stale answers with full confidence — worse than a cold start.

Neo4j replays Cypher rather than loading a dump, because `neo4j-admin load` needs the database stopped, and the platform degrades gracefully without the graph. Statements apply one at a time so a single failure does not discard the restore, and failures are counted and surfaced.

---

10. Evaluation, replay and model suitability

Added in v21.0.0, this is how the platform keeps its own numbers honest. The full methodology lives on the benchmark page; what an operator needs is below.

Serving-model attribution. Every verdict scorecard, ledger exchange and eligibility decision records which model actually served each call, with a provenance status: `gateway_verified` when the LiteLLM gateway attests the serving model in a response header, `self_reported` when only the response body names it (credited only when the body can witness itself — a name merely equal to the requested alias is never copied in), and `unattributed` otherwise. A run that placed zero real calls, fell back to the deterministic path, or cannot name its serving model is preserved as a diagnostic and refused as "verified performance of model X". The same status gates autonomy: a clearing measurement that cannot be bound to an attributed call cannot authorize automatic closures.

Metrics with denominators. The replay scorer publishes precision, recall, per-class F1, confusion matrices, balanced accuracy and MCC with numerator/denominator counts; full-cohort malicious recall keeps abstentions and runtime failures in the denominator; closure candidates carry four honest cuts (malicious closure rate, closure contamination, benign closure coverage, unknown-label share), always over proposed closures — a simulated closure never appears as a production outcome. Requested, attempted, completed, failed and unattempted counts travel with every run, so a failure that stops a run cannot vanish from the headline.

Alert reduction, three figures. 75.3% is the legacy four-tier grouping benchmark (not what production runs); 33.3% is the production correlation key on the tactic-carrying cohort; 24.5% is

a detection-shaped cohort after v21.0.0's grouping change, where a missing identifier narrows the group instead of widening it. The drop is the honest direction: the difference was unrelated alerts counted as grouped.

Replay and re-hunting. Historical replay freezes evidence at each decision time, enforces the cutoff in retrieval and storage queries rather than in the prompt, keeps truth labels reachable by the evaluator and unreachable by the agent, and discloses that pretrained weights may postdate the events (the scorecard carries the model-knowledge limit, or "unknown"). Replay cannot execute real actions — the mutation boundary is the write adapter, not an instruction. Re-hunting old telemetry with newly published intelligence is a separate, labelled mode with a rehearsal switch (`RETRO_HUNT_DRY_RUN=1`) that records proposed reopens without mutating anything; its discoveries are never counted as predictions at the original event time.

Model suitability is per task. `scripts/run_model_matrix.py --task verdict` grades each configured model on the same locked corpus and records serving identity, sampling, runtime and token use per row. Every row is read against a deterministic baseline floor (balanced accuracy exactly 0.250 on the published corpus — what answering without a model scores). A model or configuration with no funded run is listed as not measured; reproduction is one command against any OpenAI-compatible endpoint, including a local Ollama.

11. Honest limits

Stated plainly, because a guide that only lists strengths is a brochure.

- **No hosted LLM provider has been exercised.** Every published agent figure comes from a locally served model. The hosted path exists and is untested.
- **Long-duration scale figures do not exist.** The harness has ramp, burst, backpressure and 24/72-hour soak profiles; the long ones have not been run, and no throughput number is claimed from them.
- **Package distribution is half-shipped.** The five PyPI packages are live and resolve at the tree's versions (`pip install aisoc-sandbox` works today); the three npm packages still await the first-upload `NPM_TOKEN`, an account action. The release workflow builds and packs all eight on every tag.
- **Benchmark numbers are labelled.** Three of the four eval metrics are substrate self-consistency gates, not agent accuracy. Only MITRE accuracy measures the live agent, and the benchmark page says which is which.
- **Response is not autonomous by default.** It requires explicit policy authorization and a human approver.

---

12. Where to go next

You want	Read
The full documentation portal	`https://beenuar.github.io/AiSOC/`
The API specification	`docs/openapi.yaml`
What changed and when	`CHANGELOG.md`
What is gated and by which job	`docs/audit/CLAIM_TO_GATE_MATRIX.md`
To report a vulnerability	`SECURITY.md`
To contribute	`CONTRIBUTING.md`

Credits

Development of AiSOC is **funded and supported by Cyble** (https://cyble.com), who pay for the engineering time behind the project and release it under the MIT licence rather than keeping it.

That funding buys no special treatment in the product: there are no Cyble-only features, no gated modules, and no telemetry reporting back to anyone. The same code, under the same licence, for everybody.