SECURITY WHITEPAPER

Sentinel security architecture.

Written for the security team that has to sign off on an inline decision gate. It covers architecture, data flow, authentication, key management, the threat model including what we do not defend against, and how patches reach you.

Version 1.0 · 28 July 2026
Product Sentinel, by Integrators AI LLC
Deployment Self-hosted by default
Certification None held — see posture and §9

1. Summary and posture

Sentinel sits between an AI agent and the action it proposes. It evaluates the proposed action against configured rules and returns APPROVE, BLOCK, FLAG or ESCALATE before execution. It is a control in the decision path, which is a stronger claim than observability and a correspondingly larger review burden.

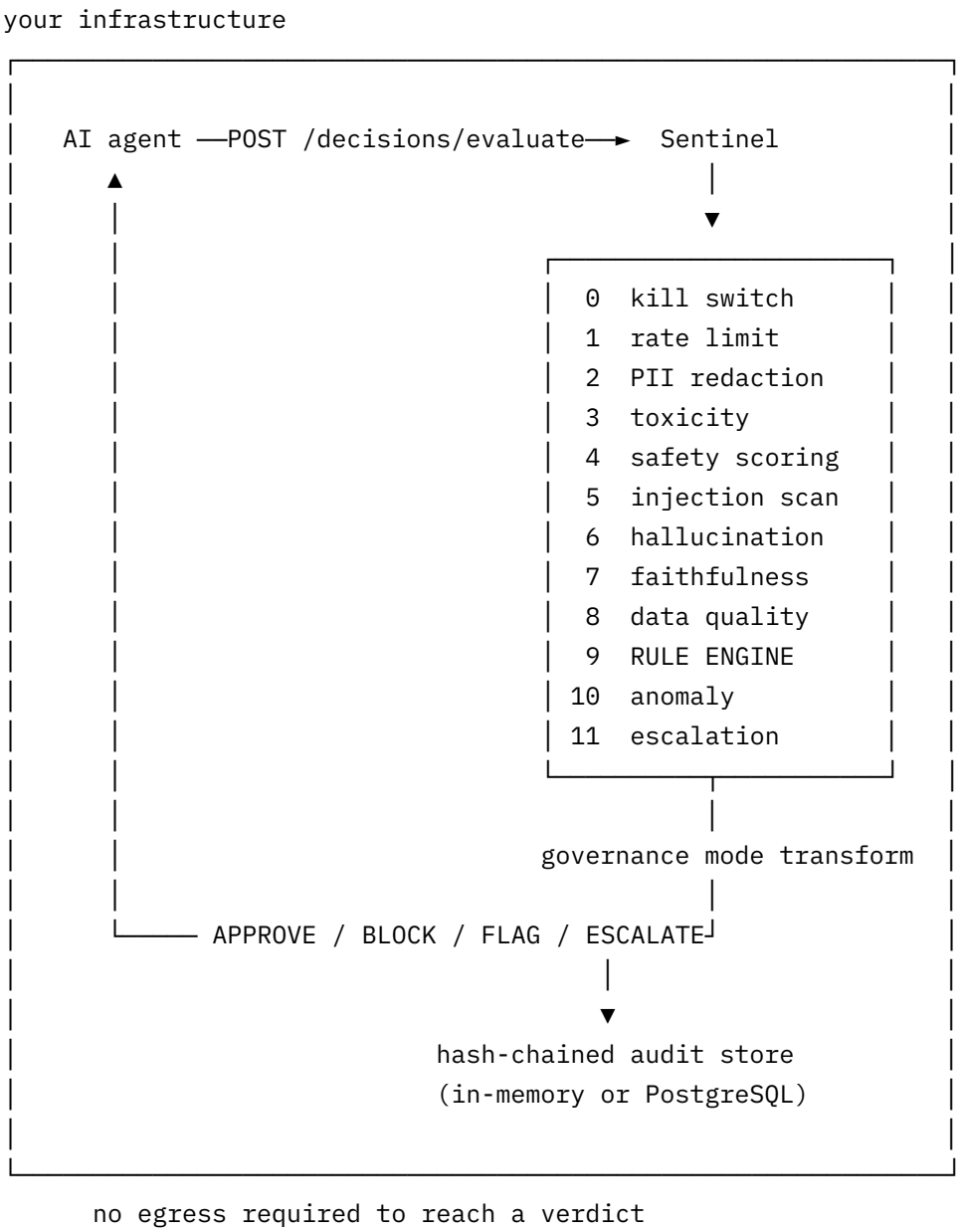
Three properties define the security posture, and each is architectural rather than procedural:

- **Self-hosted by default.** Decision data, the audit chain and evidence bundles remain inside your infrastructure. We do not receive them and have no means of retrieving them.
- **No LLM in the decision path.** The pipeline is deterministic pattern matching and rule evaluation. There is no model inference, no external API call, and no network egress required to reach a verdict.
- **Independently verifiable output.** The evidence format is published and a dependency-free verifier ships with it, so you can check our integrity claims without trusting our software. See verification.

> ### Stated plainly
>
> We hold no third-party security certification. No SOC 2, no ISO 27001, no commissioned penetration test. If a certificate is a hard procurement gate, we are not yet a fit. §9 covers what triggers the audit.

2. Architecture

A single Python service (FastAPI) with an optional PostgreSQL backend. No message broker, no external inference dependency, no third-party service required at decision time.



Stages 2 through 8 are advisory: they annotate and score. Stages 0, 1, 3 and 5 can terminate early and return BLOCK without reaching the rule engine. Stage 9 is the control that decides. See the pipeline and the measured cost of each stage.

3. Data flow and residency

In a self-hosted deployment there is no operational egress. Concretely:

DATA	WHERE IT LIVES	REACHES US?
Decision parameters	Your process memory; optionally your PostgreSQL	**No**
Audit chain and evidence bundles	Your store, append-only	**No**
Rules, thresholds, agent registry	Your store	**No**
API keys and PoA secrets	Your store, hashed (§5)	**No**
Support correspondence you send us	Our email	Yes, what you choose to send

PII redaction happens before rule evaluation. Stage 2 scans every string parameter against 24 pattern families and replaces matches with typed tokens; the redacted copy is what later stages and the audit record see. Detection is regex, so it catches the formats those patterns describe and will miss others — treat it as reducing incidental exposure, not as a guarantee that no PII is written.

Optional outbound connections, all off by default and all configured by you: webhook alerts, Jira and ServiceNow change-record posting, and an OpenAI-compatible LLM endpoint used *only* by the demo chat assistant and never by the decision path.

4. Authentication and authorisation

API-key authentication with role-based access control. Authentication is **off in development and enforced in production** (`SENTINEL_ENV=production`), which is a deliberate ergonomics choice and a configuration risk worth checking on your first deploy.

- **Keys** are presented via the `X-API-Key` header or bearer token.
- **Roles** are `admin` , `editor` , `viewer` , with granular permissions per key (for example `evaluate` , `governance.write` , `kill_switch.write`).
- **The `admin` permission bypasses granular checks.** Grant it deliberately.
- **SSO** via SAML and OIDC with role mapping is available for console access.

Asymmetric authorisation on governance changes. Moving an agent toward more enforcement requires `governance_mode.write` . Moving it *away* from enforcement additionally requires `admin` and a stated reason, because demotion silently removes a control while promotion fails loudly. Kill switches invert this deliberately: any accountable owner may pull one without approval, since stopping an agent is the safe direction.

Every governance change is written to the audit chain with the calling key's name as the actor.

5. Key management

Three secret types. All are generated with the platform CSPRNG (`secrets`) and none is recoverable in plaintext after issue.

SECRET	GENERATION	AT REST	ROTATION
API key	`sk-` + 32 bytes urlsafe (256-bit entropy)	SHA-256 hash only; shown once at creation	Create new, revoke old via `DELETE /auth/keys/{id}`
Proof-of-Agent secret	`poa-` + 32 bytes urlsafe	SHA-256 hash only; returned once	In place via `POST /agents/{id}/poa/issue`
Instance secret	Supplied by you as `POA_INSTANCE_SECRET`	Environment only; never persisted by us	Manual — change the variable and restart

Possession verification uses a constant-time comparison against the stored hash. The instance secret, when set, produces `instance_attestation = HMAC-SHA256(secret, combined_digest)` on every proof, letting you demonstrate a bundle came from your instance. A third party cannot verify that value without the secret, and it is not required for the integrity checks in the evidence spec.

Limitations, stated rather than discovered

No automatic key expiry. Keys are valid until revoked. If your policy requires maximum key age, you must enforce it operationally.

No in-place API key rotation. The pattern is create-then-revoke, which means a brief window with two valid keys.

Rotating the instance secret invalidates prior attestations. Previously issued `instance_attestation` values will no longer verify. Bundle integrity is unaffected; only the instance-provenance claim is.

We do not manage a KMS or HSM. Secrets live in your environment and your database. Integrating with a key vault is your deployment's responsibility and we

do not currently provide a driver for one.

6. Audit integrity

Every decision and governance action appends to a hash chain: `payload_hash = SHA256(canonical_json(payload))` and `chain_hash = SHA256(previous_hash | payload_hash | event_id | created_at)`. Modification of a retained event breaks the chain and is detectable by anyone holding the log.

The format and verification procedure are published in full, with a standalone verifier that imports nothing from Sentinel. That independence is deliberate: a verifier sharing our hashing code would only prove we agree with ourselves.

Tamper-evident, not tamper-proof

Hash chaining detects **modification** of retained events. It does not by itself prevent **deletion**, and an actor with write access to the store can truncate the chain. If you require deletion resistance, place the log on immutable object-lock or WORM storage. We do not claim to be WORM storage and you should not accept vendor copy that does without one behind it.

7. Threat model

Threats we considered, and honestly where each stands.

MITIGATED

Prompt injection in decision parameters

Stage 5 scans inbound parameters against seven known attack families. **Partially effective and measured:** precision 1.00, recall 0.03 against a public benchmark. See the published eval. Treat as one deterministic layer, not a perimeter; the rule engine behind it is the control that carries weight.

MITIGATED

Tampering with a decision record

Hash-chained append-only events, independently verifiable (§6). An altered payload fails verification.

MITIGATED

Forging a decision's provenance

Proof-of-Agent binds the decision to the agent identity, template version and rule set in scope, and the canonical strings are cross-checked against the decision record, so a valid proof lifted from another decision fails.

MITIGATED

Client selecting its own enforcement level

Governance mode is server-authoritative. `/decisions/evaluate` accepts no mode field. A control the controlled party can configure is not a control.

MITIGATED

A misbehaving agent needing immediate stop

Kill switches at agent, agent-type and global scope, evaluated at stage 0 before anything else and exempt from governance-mode downgrade. A stopped agent is stopped in every mode.

PARTIAL

PII leakage into logs

Stage 2 redacts 24 pattern families before evaluation and before the audit write. Regex-based, so unusual formats are missed. Reduces incidental exposure; does not guarantee absence.

PARTIAL

Denial of service by request volume

Per-agent-type rate limiting at stage 1, configurable. This is application-level only; network-layer DDoS protection is your deployment's responsibility.

PARTIAL

Malicious or mistaken policy change

RBAC, asymmetric authorisation on demotion, release-gated eval sets, and an audit event with a named actor. An operator with `admin` can still weaken policy; the control is that it is recorded and attributable, not that it is impossible.

MITIGATED

Gate unavailability treated as approval

The SDK and LangChain integration fail **closed** by default: an unreachable or slow gate raises rather than returning anything approval-shaped. Fail-open is available but must be chosen explicitly, and the response it returns reports `is_governed` as False with no proof and no audit record, so an unevaluated action cannot be laundered as an evaluated one.

NOT DEFENDED

Compromised host or database

An attacker with write access to your store can delete audit events or alter rules. Chain verification will show a break; it will not prevent the act. Host and database security are yours.

NOT DEFENDED

Novel prompt injection

Pattern matching against known shapes has a ceiling. A phrasing matching no family passes stage 5. This is the fundamental limit and no amount of pattern authoring removes it.

NOT DEFENDED

A lawful-but-wrong policy

Sentinel enforces the rules you configure. If a rule encodes a proxy for a protected characteristic, it will be enforced faithfully. Bias monitoring is designed to surface that case; the obligation remains yours.

NOT DEFENDED

Supply-chain compromise of our dependencies

We pin dependencies and keep the runtime surface small, but we do not currently produce a signed SBOM or reproducible build. Self-hosting means you control what you deploy and can inspect it.

8. What we do not defend against

Restated compactly, because this is the section a reviewer should be able to quote:

- **Novel prompt injection.** Known shapes only, at 3% recall against a public benchmark.
- **Deletion of audit records** by an actor with store write access. Tamper-evident, not tamper-proof.
- **A compromised host.** No runtime self-protection or integrity monitoring of our own process.
- **Network-layer attacks.** No DDoS mitigation, no WAF, no TLS termination in-app.
- **Bad policy.** We enforce your rules; we do not validate that they are lawful or fair.
- **Model-output filtering.** Stage 5 scans inbound parameters, not model responses to end users.
- **Multi-turn attacks assembled across requests.** Scanning is per-request and stateless.
- **Insider misuse by an `admin` key holder.** Recorded and attributable, not prevented.

9. Patching and disclosure

Distribution. Self-hosted, so you control when a change lands. There is no forced auto-update and no remote code push into your environment.

Verification. Over 2,300 automated tests run on every change, including adversarial cases and the published eval. The evidence-format tests assert that our own verifier and an independent reimplementation agree, so a change that silently alters the hashing contract fails the build.

Vulnerability reports follow the coordinated disclosure process: acknowledged within 3 business days, reproduced as a failing test, added to the eval corpus — which makes our published numbers worse until fixed — then fixed or documented as not-fixed, and re-measured.

Certification. None held. The trigger for beginning a SOC 2 observation window is the first design-partner contract; running one before a single institution has committed would spend runway on a certificate rather than the product. When it starts, dates and the auditor are published on the posture page and owned.

10. Open questions we expect

Rather than wait to be asked:

- **“What happens when the gate is unreachable?”** Configurable, and **fail-closed by default**. An unreachable or slow gate raises `GateUnavailableError` and the agent does not proceed. Callers who prefer availability can opt into fail-open, which returns a response that is deliberately distinguishable from a governed one — `is_governed` is False, there is no proof, no audit record and no `decision_id`. Timeouts and refused connections are treated as the same availability event, and the default client timeout is 5 seconds against a pipeline with a 1.3 ms p50. See verification and the SDK documentation.
- **“Can we see a penetration test?”** None has been commissioned. Internal adversarial testing exists in the suite, which is not the same thing.
- **“What is your SLA?”** Self-hosted, so availability is a property of your deployment. We do not operate the runtime and will not claim an uptime figure for software running in your data centre.
- **“Multi-tenancy?”** Single-organisation today. Not a supported isolation boundary.
- **“Who is accountable?”** One person. You will have their number, and there is no tier-one filter between you and them.

If your questionnaire covers something not addressed here, send it — we would rather answer it in writing and add it to this document than handle it verbally once.

---

contact@integrity-stack.com