AOSIQ Threat Model

What we protect against. And what we don’t.

AOSIQ provides specific, enforceable security properties — and is explicit about what it doesn’t cover. Security teams reviewing the runtime should be able to read this document and decide either way without running into marketing fog.

Field	Value
Document	v0.7.0
Last updated	May 2026
Reading time	≈ 12 min
Audience	Security teams

---

§ 01 / Summary — What this document is

This document lists every security property AOSIQ enforces, with references to the files, migrations, and audit events that implement each property. It also lists, with equal specificity, the threats AOSIQ does not mitigate.

The honest framing matters. A threat model that claims comprehensive coverage is either overpromising or has stopped thinking about new attack vectors. Real security writing acknowledges what it doesn’t cover, why, and what compensating control the operator is expected to provide.

Read alongside the file references table at the bottom — every claim made here can be verified against the codebase. **Nothing in this document is aspirational.** Items still in development are not included; they appear in the public roadmap separately.

---

§ 02 / Coverage — What’s in scope

Six classes of threat the runtime materially reduces. Each is implemented as a runtime invariant — not as an operator best practice, not as documentation, not as a configuration default.

Mitigated by runtime invariants:

- **Capability narrowing prevents agent-to-agent privilege escalation.** A child agent’s grants are the *intersection* of the parent’s grants and the child’s request. Empty intersections raise `InsufficientAuthority` at spawn time, before any code runs.
- **Per-session audit chain detects PostgreSQL-write tampering** when the audit anchor store is independently credentialed. Mid-chain row modifications break the SHA-256 linkage and fail verification.

- **Mandatory approval prevents autonomous destructive actions** when tools are registered with `reversible=False`. The runtime cannot execute the tool; it inserts a pending review row and blocks the agent until an operator approves the specific `(tool, args_hash)` pair.
- **Cost ceilings prevent runaway LLM spend.** Configured per-session, the ceiling raises an exception *before* the API call would push usage over budget — not after the bill arrives.
- **Per-agent-class backend constraints prevent unintended data egress.** An agent class registered with `allowed_backends={"ollama"}` cannot route to Anthropic, OpenAI, or any other provider regardless of operator misconfiguration.
- **Untrusted-content delimiters and pattern detection raise the floor against direct prompt injection.** Tool results are wrapped with structural markers and scanned for injection-shaped patterns; detections prepend warnings and write audit events without breaking legitimate use.

---

§ 03 / Limits — What's explicitly out of scope

Six classes of threat AOSIQ does **not** mitigate today. Each is documented because operators evaluating the runtime need to understand what compensating control they remain responsible for.

Not covered — operator responsibility:

- **Fully-compromised infrastructure operator.** An attacker holding both PostgreSQL write access and audit-anchor store credentials can rewrite the chain and the anchors together. Mitigation: route audit anchors to an external append-only service (object lock, immutable storage tier).
- **Reasoning-redirection injection.** Adversarial content designed to make the LLM reach a specific *conclusion* — rather than call a specific tool — produces structurally-valid proposals with proper citations that just arrive at the wrong answer. The judge-model defense pattern is the standard mitigation; deferred to a future release.
- **KB content poisoning at ingest time.** Documents embedded in the knowledge base are trusted at ingest; AOSIQ does not scan for prompt-injection patterns or PII before embedding. Operators are the trust boundary for what enters their corpora.
- **Multi-turn injection drifting across many tool results.** An adversary placing content across a long sequence of tool calls, gradually steering the agent's reasoning, evades single-result pattern detection. No good general defense exists; explicit deferral.
- **Adversarially-crafted patterns that evade the regex detectors.** The injection pattern set is best-effort, not exhaustive. A determined adversary can construct phrasings that look benign to the detector and still land as injection in the LLM's context.

- **LLM hallucination of correct-looking-but-wrong tool arguments where capabilities permit.** The seven-layer evidence stack catches most cases but cannot prove absence. Operators remain responsible for treating proposals as advisory, not authoritative, in scenarios where errors are costly.

---

§ 04 / Scenarios — Threat scenarios with mitigations

Each scenario describes a specific attack class the runtime encounters in production deployments. The mitigation is the concrete code path or audit event that addresses the threat; the residual is what an operator should still watch for.

Severity scale:

- **Critical** — Direct path to compromise if exploited and unmitigated. Compensating controls are the operator's responsibility and should be considered required.
- **Significant** — Material risk to integrity or availability. Mitigation reduces probability or blast radius; residual exposure remains and is documented.
- **Limited** — Operationally bounded. Mitigation is structural; residual is recoverable through standard operator response.

S-01 — Untrusted HTTP caller spawns agents with arbitrary tool grants — *Critical*

Threat. An attacker with network access to the AOSIQ API attempts to `POST /api/v1/agents` with capability grants they shouldn't possess, then directs the resulting agent to perform actions in the operator's environment.

Mitigation. Bearer-token authentication on every API route via `aos/server/auth.py`. Three modes (`enforced`, `warn`, `disabled`); production deployments require `AOS_AUTH_MODE=enforced`. The `warn` mode adds an explicit `X-AOS-Auth-Warning` response header so misconfiguration surfaces in logs. Capability grants requested at spawn are clamped to the caller's authorized token claims; a caller cannot mint a token broader than its own.

Residual. The single bearer-token model does not provide per-caller audit attribution. Multi-key auth with rotation is on the roadmap. Until then, deployments with multiple upstream callers should provision per-deployment instances rather than sharing a single token.

S-02 — Compromised tool result tampers with downstream state — *Significant*

Threat. An MCP server, file-system content, or network response returns content designed to make the LLM emit an action the operator did not authorize.

Mitigation. Three layers compose. **Capability narrowing** limits the action surface: an agent without `delete_records` in its capability token cannot call it regardless of what the LLM is talked into. **Approval gating** on irreversible tools ensures destructive actions require explicit operator sign-off at the moment of execution. **Untrusted-content delimiters** (`<untrusted_tool_output>` tags) plus pattern detection raise warnings and write `APP_INJECTION_PATTERN_DETECTED` audit events when tool output looks injection-shaped.

Residual. Reasoning redirection (steering the LLM to a wrong conclusion within its granted capability set) is not detected. Critical workflows should treat agent proposals as advisory and route final decisions through human review.

S–03 — Long-running agent's capability token expires mid-session — *Limited*

Threat. An agent waiting hours on human approval for an irreversible action holds a JWT capability token that expires before approval arrives. The agent fails at resume time and the workflow stalls.

Mitigation. Token TTL is configurable via `AOS_TOKEN_TTL_SECONDS` (default 3600). Long-approval-cycle agent classes should be deployed with longer TTLs. The scheduler's heartbeat-and-reaper flow detects expired-token agents and surfaces them as recoverable rather than terminal.

Residual. No automatic token refresh on dispatch. Agents whose TTL expires mid-execution require operator intervention to re-mint. Refresh-on-dispatch is a roadmap item.

S–04 — Worker crash leaves agents stuck in `running` state — *Limited*

Threat. A scheduler worker crashes after claiming an agent but before completing its step. Without cleanup, the agent's row in `agent_processes` stays in `running` indefinitely and is not re-queued.

Mitigation. Worker heartbeat (migration `015`) records `last_heartbeat` on every agent claim. An orphan reaper loop detects agents whose worker has gone silent past a threshold, transitions them back to `runnable`, and re-queues them. Combined with the three-part composite checkpoint, the agent resumes cleanly from its last successful step. Tool-call idempotency on the agent's tool implementations is the expected compensating control.

Residual. At-least-once tool execution is possible if the worker crashes after a tool call started but before the result was recorded. Agent tool implementations are expected to be idempotent; this is documented as an integration requirement.

S–05 — Operator approves the wrong action — *Significant*

Threat. The approval gate is only as strong as the operator reviewing the prompt. An operator approving rapidly, under pressure, or without reading the proposed arguments can authorize a wrong or harmful

action.

Mitigation. The dashboard renders proposed action, full arguments, blast radius classification, evidence trail, and capability scope at the moment of decision. Approvals are bound to the specific `SHA-256(canonical_json(args))`; modifying the args after approval invalidates the approval. Audit chain captures the approving operator's token `jti` for forensic attribution.

Residual. Approval fatigue is an operational risk that grows with volume. Operators should configure capability templates tightly enough that high-volume actions are `reversible=true` and only genuinely destructive actions reach the queue.

S–06 — Audit anchor store goes silent — *Significant*

Threat. MinIO or the configured S3-compatible anchor store becomes unreachable. Without the operator noticing, the tamper-evidence guarantee silently degrades — new anchors stop being written, and an attacker with PG write access has a longer window to rewrite chain rows before detection.

Mitigation. The runtime tracks anchor write success and failure counters at the process level; both are surfaced in `/api/v1/health` as `audit_anchors`. Operators are expected to alert on any failure count above zero. Anchor write attempts are non-blocking — chain integrity continues even if anchoring fails — but the gap is visible.

Residual. Operators who do not alert on `audit_anchors.failures` can run for arbitrary periods with degraded tamper evidence. The metric is available; operational discipline to monitor it is the compensating control.

---

§ 05 / Roadmap — What's coming next

The threat model is a living document. The mitigations below are explicitly out of scope today and on the active roadmap. Each is sized, scoped, and has a target release. Threat scenarios graduate from this list to the in-scope section as mitigations ship.

ID	Mitigation	Status
R–01	Judge-model pattern for reasoning-redirection injection	Next release
R–02	KB ingest scanner — pre-embedding scan for injection, secrets, PII	Next release
R–03	Multi-key authentication with rotation and per-caller audit attribution	Researching
R–04	OIDC and mTLS for production deployments	Researching

R–05	Performance characterization under realistic concurrency	Researching
R–06	Sandboxed code execution as a native tool	Next release

R–01 — Judge-model pattern. High-severity proposals routed to a second LLM call with only the proposal and evidence — no original task, no tool history. Judge disagreement surfaces to operator. Closes the largest open category in the prompt-injection threat surface.

R–02 — KB ingest scanner. Pre-embedding scan for prompt-injection patterns, secrets, PII, and known-malicious content. Operators see a flagged list and approve before ingest. Closes the KB-poisoning gap documented in § 03.

R–03 — Multi-key authentication with rotation. Per-caller bearer tokens with independent audit attribution and rotation without downtime. Resolves the residual exposure noted in scenario S–01 for deployments with multiple upstream callers.

R–04 — OIDC and mTLS for production deployments. Federated identity for API callers, mutual-TLS as an alternative bearer-token mechanism. Targets enterprise deployment shapes where bearer tokens alone are insufficient for the security review process.

R–05 — Performance characterization under realistic load. Published p50/p99 latency numbers, soak test results, capacity planning guidance under realistic concurrency. Required for first-customer production deployment at scale.

R–06 — Sandboxed code execution as a native tool. Container-isolated Python execution with no network, ephemeral filesystem, and resource limits. Replaces over-broad bash grants with a tighter primitive. Reversible by construction; no approval gate required.

Items marked *deferred* are intentional gaps where no good general defense exists today (multi-turn injection drift, adversarial pattern evasion). These remain operator-responsibility until the security research community converges on accepted mitigations.

---

§ 06 / Operator — Operator responsibilities

Several mitigations require operator action to remain effective. These are not features the runtime enforces — they are conditions the runtime depends on.

- **Run with `AOS_AUTH_MODE=enforced` in production.** The `warn` mode is for development. The `X-AOS-Auth-Warning` response header makes misconfiguration visible in logs and reverse-proxy traces.
- **Credential the audit anchor store independently from PostgreSQL.** Tamper evidence collapses if both stores are reachable from the same compromise. Anchor credentials should

live in a separate secrets path with separate rotation.

- **Treat agent proposals as advisory in critical workflows.** The seven-layer evidence stack catches most hallucination cases; it cannot prove absence. Production deployments where errors are costly should keep humans authoritative on outcomes.
- **Vet KB ingest sources.** Documents added to the knowledge base are trusted by the agents that retrieve them. Ingest from sources you control, or scan documents externally before embedding.
- **Alert on anchor write failures.** The `/api/v1/health` endpoint surfaces `audit_anchors.failures`. Any non-zero value indicates the tamper-evidence guarantee is degrading.
- **Configure capability templates restrictively by default.** The runtime cannot prevent over-broad grants if the operator authorizes them. Build agent classes with the smallest capability set that completes the task.

---

§ 07 / References — File and migration references

Every claim in this document maps to specific files, migrations, or audit event types in the codebase. Operators with source access can verify each property directly.

Reference	Property
`aos/api/syscalls.py`	Single policy enforcement boundary; capability check, audit append, approval gate, cost ledger record fire here
`aos/security/capability.py`	JWT capability token mint, verify, delegate (intersection-narrowing)
`aos/audit/engine.py`	Per-session SHA-256 hash chain; anchor write to MinIO/S3
`aos/server/auth.py`	Three-mode bearer-token authentication scaffold
`aos/kernel/scheduler.py`	Worker heartbeat, orphan reaper, state-machine validated transitions
migration 012	Per-session audit chain with `chain_seq` and `pg_advisory_xact_lock`
migration 014	State-machine validation function; invalid transitions match zero rows
migration 015	Worker heartbeat columns and orphan-detection index
`APP_INJECTION_PATTERN_DETECTED`	Audit event written when tool output matches injection-shaped patterns
	Audit event written when an

`HUMAN_APPROVAL_REQUESTED`	agent attempts an irreversible tool
`APP_AGENT_ABANDONED_UNGROUNDED`	Terminal state when an agent refuses to investigate before proposing
`/api/v1/health`	Returns auth_mode, audit_anchors success/failure counts, scheduler status

§ 08 / Log — Changelog

A maintained threat model has a changelog. New mitigations, new threat scenarios, and reframings are recorded here.

Date	Version	Change
May 2026	v0.7.0	Added scenario S–02 covering prompt-injection in tool outputs; documented untrusted-content delimiter mitigation and `APP_INJECTION_PATTERN_DETECTED` audit event. Added severity ratings (Critical / Significant / Limited) on all scenarios. Added § 05 Roadmap section with six in-flight mitigations.
Apr 2026	v0.6.0	Clarified S–06 (audit anchor degradation); added `audit_anchors` counters to `/api/v1/health`; documented operator-monitoring expectation.
Mar 2026	v0.5.0	Added scenario S–05 (operator approval mistakes); bound approvals to `SHA-256(canonical_json(args))` for argument integrity.
Feb 2026	v0.4.0	Orphan recovery via heartbeat and reaper (migration 015) added; S–04 documented.
Jan 2026	v0.3.0	Per-session audit chains (migration 012); chain serialization moved from global to per-session via `pg_advisory_xact_lock`.

Contact

Security disclosure: security@aosiq.com — see `/.well-known/security.txt` for the full policy and SLA.

General contact: hello@aosiq.com

Web: https://aosiq.com/threat-model.html — always-current version with two-column layout and active-section highlighting.