

# Aphoria

## A code-level truth linter powered by Episteme.

Aphoria scans a codebase, extracts the decisions embedded in config and code, and checks them against authoritative sources. It finds the places where what your code *does* contradicts what the specs *say*.

---

## The Problem

Every codebase contains implicit claims. `verify=false` is a claim that TLS verification isn't needed. A JWT library configured without audience validation is a claim that aud checking doesn't matter. A hardcoded timeout of 0 is a claim that requests should wait forever.

These claims are invisible. They live in config files, middleware setup, environment variables, and constructor arguments. Nobody thinks of them as claims. They're "just code."

But they conflict with authoritative sources. RFC 7519 says audience validation is mandatory. OWASP says TLS verification must be enabled. Vendor documentation says the default timeout exists for a reason.

Today, the gap between what the code does and what the specs say is discovered in one of three ways:

1. **A human reviews it.** Expensive, inconsistent, doesn't scale.
2. **A linter catches it.** Only works for known patterns. Can't reason about intent.
3. **Production breaks.** The \$180M way.

AI agents make this worse. An agent deploying code doesn't read the RFC. It picks the most confident-sounding answer from its training data — or the most recent Stack Overflow snippet — and acts. The agent doesn't know it's contradicting a standard because it has no structured way to check.

## The Solution

Aphoria gives codebases an epistemic audit trail.

```
$ aphoria scan ./citadeldb
```

```
Scanning citadeldb (rust) ...
```

```
code://rust/citadeldb/auth/jwt/audience_validation
  Your code: aud validation DISABLED (src/auth/jwt.rs:47)
  RFC 7519:  aud validation MUST be enabled (Tier 0, confidence 1.0)
  Conflict:  0.92 - BLOCK

code://rust/citadeldb/net/tls/cert_verification
  Your code: certificate verification DISABLED (src/net/client.rs:23)
  OWASP:     certificate verification REQUIRED (Tier 1, confidence 0.95)
  Conflict:  0.87 - BLOCK

code://rust/citadeldb/http/timeout
  Your code: timeout = 0 (no timeout) (config/production.yaml:8)
  Vendor:    default timeout 30s recommended (Tier 2, confidence 0.7)
  Conflict:  0.54 - FLAG
```

```
3 conflicts found (2 BLOCK, 1 FLAG)
```

Aphoria doesn't lint syntax. It lints *epistemic drift* — the gap between what your code asserts and what authoritative sources say.

## How It Works

1. **Walk the project.** Map directory structure to ConceptPaths under code://.
2. **Extract claims.** Pattern-based extractors find security configs, dependency versions, timeout values, auth patterns.
3. **Ingest into Episteme.** Each extracted claim becomes an Assertion with the project's code as the source (Tier 3).
4. **Check for conflicts.** Query Episteme for each concept. If authoritative sources (Tier 0-2) disagree with the code claim, the conflict score fires.
5. **Report.** Output the conflicts with source references, conflict scores, and escalation verdicts.

The concept hierarchy is the backbone. `code://rust/citadelldb/auth/jwt/audience_validation` automatically aliases to `rfc://7519/jwt/audience_validation` because they share the `jwt/audience_validation` leaf. The conflict becomes structurally visible without anyone manually connecting the two.

## Who This Is For

**Engineering leads** who deploy AI agents and need a pre-flight check. “Before the agent merges this PR, did it contradict any RFCs?”

**Platform teams** building internal developer tooling. Aphoria integrates into CI as a step between lint and deploy.

**Security teams** who audit configs across multiple services. “Across all our projects, which ones skip certificate verification?”

## What This Is Not

- **Not a linter.** Linters check syntax rules. Aphoria checks claims against external authoritative sources.
- **Not a SAST tool.** SAST finds vulnerability patterns. Aphoria finds where code decisions contradict standards, which is a superset.
- **Not a replacement for code review.** It augments review by surfacing conflicts that humans miss because they haven't memorized every RFC.

## The Skill Integration

Aphoria ships as both a CLI and a Claude Code skill. When working in a project:

```
/aphoria scan
```

The skill runs the CLI, ingests claims, queries for conflicts, and reports inline. The developer fixes the conflict or explicitly acknowledges it — which creates a new assertion: “engineering team decided to skip aud validation for internal services” (Tier 3, Expert). Now the disagreement is structured, documented, and visible next time anyone touches that code.

The acknowledge flow is important. Not every conflict is a bug. Sometimes the code is right and the RFC is too strict for the context. Aphoria doesn't force compliance. It forces *visibility*. The decision to deviate from a standard becomes a recorded, auditable, queryable fact — not an invisible default.

## The Flywheel

Every project Aphoria scans adds claims to Episteme. Every acknowledged deviation adds structured context. Over time:

- Common false positives get suppressed (the alias “internal services can skip aud” gets registered across projects)

- Common true positives get elevated (the same JWT misconfiguration across 50 projects becomes a systemic signal)
- The authoritative source corpus grows (new RFCs, new OWASP entries, new vendor docs get ingested by research agents triggered by gaps)

The more projects Aphoria scans, the smarter it gets — not through ML, but through accumulated structured disagreement.