Sentinel

Technical Whitepaper

Evidence-driven security research for explicitly authorized source repositories

YeleTech Security

Document version: 1.7 | November 15, 2026

Implementation baseline: Sentinel 0.1.0, commit `86fc22409a3e897c2976a7f19ccec496afeae031`

Revision scope: frozen evaluation protocols, coverage accounting, local adjudication and held-out pilot validation; repository links use local relative paths.

Status: local source scans, operation attestations and bounded follow-up implemented; broader research platform in development.

Contents

Abstract 3

1. Purpose, scope, and evidence conventions 3

2. Problem formulation and design principles 4

3. Current capability boundary 5

4. Architecture and data flow 6

5. Canonical contracts and finding semantics 7

6. Engagement policy and bounded execution 8

7. Pinned source and provenance 9

8. Upstream integration and containment 10

9. Source-grounded research compilation 11

10. Ranking and bounded investigation planning 14

11. Evidence integrity and auditability 15

12. Reports and operator experience 15

13. Operating a local assessment 17

14. Validation and experimental evidence 20

15. Assurance limits and residual risks 23

16. Planned platform extensions 24

17. Engineering tradeoffs 25

Appendix A. Module and source map 26

Appendix B. Document provenance and regeneration 26

References 27

Abstract

Sentinel is a security-assessment system that combines an upstream source-review runtime with a deterministic authorization layer, typed evidence contracts, source-grounded research models, and an offline review interface. Its central design principle is that a tool or model may propose a security claim, but the claim's provenance, execution authority, and verification status must be represented and checked separately.

The implemented system scans an explicitly authorized local Git repository at a pinned commit, confines the upstream scanner to a bounded Docker environment, restricts network egress to a configured model-provider endpoint, and preserves results as unverified candidates. A separate deterministic research pipeline interprets supported Python HTTP/header-to-SQLite patterns, follows bounded local call chains, and connects proposed hypotheses to protected assets, principals, trust boundaries, invariants, and hashed source evidence. Reviewers receive both machine-readable artifacts and self-contained HTML reports.

This paper describes the architecture, contracts, algorithms, operational workflow, validation evidence, and trust assumptions of the original baseline and the explicitly identified working-tree extension. It distinguishes implemented behavior from proposed capabilities, including general independent verification, causal vulnerability chaining, incremental research, and authenticated multi-tenant operation. The available evidence demonstrates local workflow integration and negative-control behavior; it does not establish general vulnerability-discovery accuracy or production readiness.

1. Purpose, scope, and evidence conventions

Sentinel addresses a practical gap between receiving a scanner finding and establishing a defensible security conclusion. A useful assessment must answer several different questions: what source was examined, what action was authorized, what the scanner actually reported, what security property may be violated, which assumptions support that interpretation, and what experiment could confirm or disprove it. Combining those questions into a single model-generated verdict makes review difficult and can overstate assurance.

Sentinel separates them into independently inspectable records. The current product is a local, single-operator source-review workflow. It is intended for repositories that the operator is authorized to review and supply to the configured model provider. It does not provide internet-wide discovery, autonomous production exploitation, credential collection, persistence, destructive testing, or production lateral movement. These exclusions follow the build specification's scope. [R1]

Three implementation-status terms are used throughout this paper:

- **Implemented:** behavior present in the baseline or identified extension, supported where stated by repository tests or recorded runs.
- **Limited:** an implemented behavior with a deliberately narrow scope or a material operational qualification.
- **Planned:** a contract, requirement, or design direction whose complete execution path is not implemented.

The implementation, decision log, and backlog take precedence when describing current behavior. The build specification describes the intended platform and contains commands and architecture sketches that are not all implemented. Repository references at the end of this paper provide the supporting primary sources. Reported experiment costs and model identifiers are historical observations, not pricing guidance or recommendations. [R1-R4]

2. Problem formulation and design principles

2.1 Claims, evidence, and authority are different objects

A scanner can produce a plausible narrative without reproducing the claimed behavior. A source citation can identify the correct lines without proving reachability. A reproduced behavior can remain outside the operator's authorized scope. Sentinel therefore treats claim quality, evidence identity, and execution permission as separate dimensions.

The principal design rules are:

1. **Authorization precedes execution.** Deterministic policy checks authorize Sentinel-managed actions; a model response cannot expand scope.
2. **Source identity precedes interpretation.** Research citations are checked against committed bytes, not the current working tree.
3. **Upstream verdicts remain upstream claims.** A scanner's `verified` field does not become a Sentinel-confirmed finding.
4. **Security properties are explicit.** Hypotheses refer to an invariant, an actor, a protected asset, a boundary, and required evidence.
5. **Unknowns are retained.** Unsupported code and unresolved assumptions produce gaps rather than fabricated architecture.
6. **Reuse is the default.** Existing runtimes perform generic scanning and agent orchestration; Sentinel adds authorization, semantics, provenance, and reviewer-facing assurance.

These rules determine module boundaries and report vocabulary. They are not a claim that every intended platform control is complete. In particular, the current policy engine does not intercept each internal tool invocation made by the upstream runtime. Container restrictions and launch-level authorization constrain that runtime as a whole. [R1, R3, R6, R7]

2.2 Why a framework-first implementation

Sentinel integrates the pinned 0.security CLI rather than implementing another general agent loop. Python owns the control plane, Pydantic contracts, policy, evidence handling, and research compiler. The upstream runtime remains a separate implementation behind CLI and JSON boundaries.

This is an explicit resolution of conflicting language guidance in the original specification: decisions D-001 and D-002 retain a single Python distribution and preserve cross-language integration through versioned contracts. There is no dependency on importing internal TypeScript modules into Python. PentAGI was evaluated as an optional dynamic backend but has no current Sentinel adapter. Its addition remains conditional on measured incremental value. [R3, R5]

3. Current capability boundary

Capability	Status	Practical meaning
Local pinned-source scan	Implemented	Manifest, readiness checks, isolated scan, candidate collection, and report export.
Policy and lifecycle controls	Implemented, limited	Deny-by-default actions; scope, time, spend admission, concurrency, cancellation, and audit within an invocation.
General candidate report	Implemented	Independent of whether Sentinel understands the repository's architecture pattern.
Threat-driven assessment	Implemented, limited	Explicit context, bound operation mappings, bounded ownership proposals and revisioned reviewer decisions; independent acceptance remains pending.
Assessment plan and gate	Implemented, limited	Operation/invariant attestations, explicit scope, portable revisions and bounded navigation follow-ups; semantic acceptance is not automated.
Automatic research extraction	Limited	Python HTTP/header identity flowing into SQLite through supported direct or forwarding calls.
Linked research review	Implemented	Assets, principals, boundaries, invariants, hypotheses, citations, gaps, and a bounded planning queue.
Repository evaluation	Implemented, limited	Frozen protocols, separate labels, coverage accounting and local adjudication; independent benchmark acceptance remains pending.
Independent vulnerability verification	Planned	Schema guards exist; generic case-pack replay exists; independent oracle and finding acceptance remain pending.
Causal vulnerability chaining	Planned	Data contracts exist; typed matching and chain validation remain future work.
Incremental graph refresh	Planned	Stable scoped identities exist, but cross-commit reconciliation and retesting do not.
Authenticated hosted service	Planned	FastAPI exposes health/readiness; tenant-authenticated assessment endpoints are absent.
Persistent investigation history	Planned	PostgreSQL audit support exists; scan artifacts and research are currently local files.

Broader source scanning and narrow research extraction are deliberately separate. A repository can yield useful upstream candidates even when the research extractor returns no supported model. Conversely, a linked hypothesis is a proposal for investigation, not a finding that has been independently verified. [R2-R4, R8-R11]

4. Architecture and data flow

4.1 Implemented components

The operator CLI is built with Typer. Pydantic v2 supplies validated contracts. Git object reads establish immutable source identity. Docker supplies the upstream execution boundary, with a separate Squid proxy controlling outbound connections. A Python service assembles scan artifacts and invokes the research compiler when usable upstream source locations exist. Reports are generated as local files and require no running web service. [R6-R12]



Figure 1. Current local scan path. The research branch is optional; neither branch performs independent replay verification.

4.2 Responsibility boundaries

The upstream runtime owns its scanning strategy, agent loop, model interaction, and raw report. The adapter owns process lifecycle, capability declarations, containment configuration, status translation, artifact collection, and candidate normalization. The policy engine owns decisions on requests presented to Sentinel. The research compiler owns source validation, typed graph assembly, deterministic ranking, and review artifacts.

This division keeps upstream replacement possible without changing the meaning of a Sentinel finding. An alternative adapter must translate its output into the same contracts and preserve the distinction between availability, supported capabilities, execution outcome, and evidence strength. It cannot substitute a narrative for an unsupported replay capability. [R6, R7]

4.3 Trust boundaries

The system has four material boundaries. First, a trusted local operator supplies an engagement manifest; the manifest records scope but is not a signed authorization document or an identity-verification service. Second, target source and scanner output enter as untrusted data. Third, Docker and the host daemon enforce process and mount restrictions. Fourth, source context crosses to OpenRouter and the selected model provider during upstream analysis.

The offline report eliminates a hosted application requirement for reading results. It does not make the scan itself offline. Provider processing of source is a real data flow and must be acceptable for the selected repository. [R2, R6, R12]

5. Canonical contracts and finding semantics

5.1 Record envelope and interoperability

Canonical records carry a UUID, tenant identifier, optional project identifier, creation/update timestamps, schema version, and provenance. Provenance can identify its source, upstream version, model, and configuration digest. Models reject unknown fields and use frozen instances; these choices reduce accidental schema drift but do not constitute cryptographic authenticity or deep immutability of every nested value.

The original baseline exported 27 JSON Schemas; assessment, behavioral replay, explicit threat-model intake, threat-driven review and repository evaluation add seventeen contracts, for 44 under `contracts/v1/`. The exporter declares JSON Schema draft 2020-12, derives schemas from Pydantic, and includes a schema-version marker. Tests compare generated output with committed schemas. Breaking interface changes require a new major contract directory. [R8, R13]

5.2 Entity groups

Group	Representative records	Role
Scope	Tenant, Project, Engagement, EngagementManifest, Approval	Identify context and declared permissions.
Source and architecture	RepositorySnapshot, SystemModel, Asset, Principal, TrustBoundary, Invariant	Describe the analyzed source and security model.
Research	Hypothesis, Investigation, ReviewInput, ResearchReview, ExtractionResult	Preserve proposed claims, linked traces, gaps, and plans.
Threat context	ThreatModelInput, ThreatModelReport	Preserve supplied assumptions, provisional inferences, reference checks and unresolved gaps.
Results and evidence	Finding, EvidenceArtifact, NormalizedFinding, RunRecord	Separate canonical state from upstream claims and artifacts.
Future correlation/evaluation	CausalLink, CausalChain, EvaluationRun	Define interfaces for work not yet fully implemented.
Local scan	ScanSnapshot, ScanResult	Describe supplied source, omissions, outcome, candidates, and enrichment.

Contract availability is not equivalent to workflow availability. For example, `CausalChain` can represent a proposed chain, but the current system does not construct and verify chains through a causal-analysis engine.

5.3 Finding lifecycle and proof guard

The canonical vocabulary supports the intended lifecycle:

```
candidate -> investigating -> provisionally_supported
          -> independently_verified -> confirmed
          -> remediated -> regression_verified

Alternatives: disproved, inconclusive, duplicate,
              accepted_risk, out_of_scope
```

Figure 2. Contract vocabulary, not a fully implemented transition engine.

For `independently_verified` and later proof-requiring states, schema validation requires evidence identifiers, an independent-verifier identifier, and either a reproduced replay with listed checked assertions or a documented manual-verification exception. An explicitly supplied replay with verdict `skipped`, `error`, or `not_reproduced` cannot support such a state, even if a manual exception is also supplied. [R8]

This guard prevents structurally unsupported promotions through validated construction. It does not execute a replay, authenticate a verifier, inspect the referenced evidence, or prove that an assertion was actually checked. Those responsibilities belong to the planned verification workflow. The adapter currently creates only `candidate` findings, retains upstream status and severity as separate metadata, and leaves canonical finding severity unset. [R6, R7]

6. Engagement policy and bounded execution

6.1 Manifest-driven authorization

An engagement manifest declares its identifier, owner, expiry, repositories and commits, optional disposable environments, allowed and forbidden actions, budgets, and approval categories. YAML parsing uses a safe loader and typed validation. Contradictory allowed/forbidden actions are rejected.

The policy engine rejects unknown or unallowed actions, expired or revoked engagements, exhausted admission budgets, paths outside authorized repositories, invalid environment/network requests, and unavailable sandbox slots. Target paths are resolved before containment checks. Pinned research additionally requires the exact authorized Git root and declared commit. An allow decision is audited before it is returned; if audit persistence raises an error, the caller must not continue. [R9]

Each allowed policy request consumes a tool-call allowance; denied requests do not. Requests that reserve a sandbox consume a concurrency slot until release. These counters apply to actions presented to Sentinel. They are not a complete count of every command or model tool call inside the upstream scanner.

6.2 Approval semantics and limitations

Requests for network-enabled tests require both an allowlisted environment host and a valid approval with an approver, matching engagement, approved decision, and unexpired timestamp. High-resource requests also require approval. The manifest's `approval_required` field is accepted for compatibility, but the full intended configurable semantics and expanded-scope handling are not implemented. [R3, R9]

Model-provider egress for source review is configured by the adapter's allowlist. It is distinct from a `network_enabled_test` request against an assessment environment. The existence of model egress does not authorize dynamic testing of a remote target.

6.3 Time, cost, and interruption

The adapter refuses a requested run ceiling above the policy instance's remaining model allocation. It supplies that ceiling to the upstream scanner. The runtime deadline is bounded by both the request timeout and remaining engagement time; expiry and revocation are checked again after proxy setup and during polling.

Cost enforcement has an important limit: the adapter records upstream-reported estimated cost at run completion. It does not independently meter each provider request or reserve shared funds across processes. Upstream in-flight requests can exceed a nominal ceiling, as observed in the bake-off. A reported cost is therefore an accounting observation, not a hard billing guarantee. Non-finite or otherwise invalid upstream cost values are not accepted as valid costs. [R5-R7]

Ctrl-C and SIGTERM trigger cancellation in the local scan path. Available partial findings and lifecycle records are retained. Failure to terminate the scanner is surfaced as a failure rather than successful cancellation. Proxy/network teardown failures are audited; an operator may still need to inspect Docker after cleanup problems. Revocation enforcement depends on polling, so integrations using the adapter directly must continue to poll running work. A host crash or forced process kill is outside graceful-cancellation guarantees.

7. Pinned source and provenance

7.1 Snapshot construction

`scan init` resolves the local Git HEAD to a full commit and creates a source-only engagement. `scan run` reads the manifest pin from the Git object database and creates a separate text snapshot. Dirty working-tree changes, untracked files, and `.git` contents are not supplied to the scanner. Symlinks and submodules are not followed.

The Git reader removes inherited `GIT_*` overrides, disables replacement objects, disables lazy fetching and interactive prompting, and uses fixed subprocess arguments without a shell. Missing objects fail locally. Target code is neither imported nor executed by the host-side extraction process. These controls establish the source bytes under review; they do not make an arbitrary local repository or host immune to every Git or operating-system vulnerability. [R10]

7.2 Explicit resource and coverage bounds

Limit	Local scan snapshot	Research source reader
Source inventory	Up to 5,000 selected committed entries	Up to 100 files and bounded path probes
Per-file size	1,000,000 bytes	1,000,000 bytes
Total loaded bytes	100,000,000 bytes	8,000,000 bytes
Git command timeout	10 seconds	10 seconds
Accepted source	Regular UTF-8 text without NUL bytes	Regular committed text with valid citations

The scan omits oversized, binary/non-UTF-8, symlink, and submodule entries and records reasons. It also excludes `.env` and `.env.*`, except `.env.example` and `.env.sample`. This rule is not a general secret scanner: committed credentials elsewhere can still be supplied to the model. Selecting a prefix with no usable text is an error. A narrower prefix is the intended way to reduce source scope for a large repository. [R2, R10]

The research reader has tighter limits because it follows selected evidence, not an entire repository. Unsupported source, missing optional dependencies, ambiguous imports, and budget exhaustion must be interpreted according to their recorded gap or error, not as evidence that the code is safe.

7.3 Distinct hashes establish distinct identities

Sentinel records several digests with different meanings:

- The **policy digest** identifies the serialized enforced manifest.
- The **target digest** identifies the mounted path/content inventory.
- **Per-file SHA-256 values** identify source bytes and support citation validation.
- **Artifact hashes** identify collected tool outputs at a recorded lifecycle point.
- The **configuration digest** identifies captured adapter arguments or research configuration, according to the record that contains it.
- A research **snapshot manifest hash** covers cited files, not necessarily the complete repository or complete scan snapshot.

The host retains the authoritative scan snapshot metadata separately from the scanner's writable output. A scanner-edited copy cannot silently replace that host-held record. The adapter also verifies a completed report's digest before using its locations for research handoff. Imported saved reports remain unbound navigation hints even when their own file digest is recorded. [R6, R10, R11]

The pinned-snapshot guarantee applies to the local `scan run` workflow. Direct library users can configure the adapter with `source_prefix=None`, which retains its earlier directory-mount behavior; such usage must not be described as automatically pinning source.

8. Upstream integration and containment

8.1 Adapter contract and version pins

The adapter interface includes health, capabilities, start, poll, cancel, artifact collection, and finding normalization. Source review and cancellation are supported. Dynamic testing, runtime verification, and event subscription are reported as unsupported.

The selected upstream is 0.security v0.28.4 at commit `83305a098508992c7d3a92387c114d0664b6899f`. The default scanner image is the locally built `sentinel-bakeoff/0:v0.28.4`. The source commit is pinned, but that local image reference is a tag rather than a cryptographically attested image digest. The Squid proxy is pinned by image digest. Dependency/image review and ADR sign-off remain open. [R5-R7, R14]

8.2 Process and filesystem restrictions

The scanner receives a read-only target mount and read-only root filesystem, dropped Linux capabilities, `no-new-privileges`, temporary writable filesystems, and a designated output mount. Defaults allocate four CPUs, 4 GB of memory as expressed by Docker's `4g` option, and a 256-process limit. No host Docker socket is mounted. The image's user configuration is relied upon for its execution user; the adapter does not itself add a `--user` override. [R6]

The output directory is writable by the scanner and must be treated as untrusted. Raw reports, logs, and databases can contain model-controlled text. The reader rejects symlinked or oversized `report.json` files, with a 20 MB report-read cap. This is not a global disk quota for all scanner output. Docker, its daemon, the host filesystem, and the locally installed image remain trusted parts of the execution environment.

8.3 Egress design

Each run receives an internal Docker network without direct external routing. A proxy joins that network and supplies the allowed outbound path. Its configuration permits only HTTPS `CONNECT` requests to configured hostnames on port 443; the default hostname is `openrouter.ai`. Other methods, ports, and destinations are denied. The upstream Node runtime is configured to honor the proxy environment. [R12]

TLS is tunneled rather than inspected. The proxy restricts destinations, not semantic content, provider retention, or the underlying model provider's handling of source. It is not a data-loss-prevention system. Denied destinations are extracted for audit while paths and queries are excluded from those normalized events; the raw proxy log is retained separately and requires appropriate handling.

An earlier live run recorded blocked requests to GitHub, OSV, and NVD while model calls succeeded. That demonstrates the tested egress path and also explains a coverage consequence: external advisory lookups may be unavailable under the default allowlist. [R3]

9. Source-grounded research compilation

9.1 Two input paths

The research compiler accepts structured reviewer-authored `ReviewInput`, or input produced automatically from upstream report locations and pinned source. Structured input describes components and interfaces, principals, assets, boundaries, invariants, hypotheses, transitions, and unknowns. Automatic extraction currently derives a smaller model from supported source patterns. Both paths converge on the same validation and graph assembly. [R11, R15]

Upstream `reviewAnnotation` locations are used as navigation hints. They do not authorize arbitrary filesystem reads and do not import a scanner's proof status. Paths are constrained to repository-relative locations; citations are rebuilt from the authorized commit and checked for valid line ranges and whole-file hashes.

9.2 Explicit threat-model input

The separate optional `ThreatModelInput` accepts supplied security context without requiring invented citations. It binds a full repository commit and describes components, interfaces, assets, principals, attacker capabilities, boundaries, authentication context, invariants, oracles, assumptions and unknowns. Partial models are valid; missing context remains explicit. Source-grounded `ReviewInput` requirements remain unchanged.

Operators supply `scan run --threat-model`, or attach a local JSON file and its exact-byte SHA-256 to the engagement manifest. Mismatched commits, altered attachments and conflicting overrides prevent scanner

launch. `threat-model init` creates an editable skeleton; `threat-model check` checks references and exports an offline report without calling a model or executing target code. Context does not modify authorization.

Claim evidence labels source references as context, supporting, or contradictory, with a rationale. These labels remain supplied assertions. Deterministic checks validate file hashes and line ranges against committed source; stale references, reported contradictions and excluded context remain visible. The scanner receives a bounded, explicitly untrusted summary, while complete input and canonical digests are retained in scan artifacts and audit provenance.

Without supplied input, supported extraction creates a provisional model; unsupported source leaves architecture unknown. Supplied assumptions are preserved and source-derived inferences are stored separately for comparison. Browser/JSON exports and portable offline reassessment retain the model and its gaps. The intake's semantic status remains `not_verified`; a separate assessment records later evidence proposals and reviewer decisions without rewriting context. Version-3 assessment plans bind the model to an operation catalog and combine threat obligations with source-review readiness. Historical version-2 receipt digests and gate semantics remain valid.

Mappings identify an invariant, operation subjects, entry points, callers and required context, bound to exact model/catalog digests and source commit. Exact keys, matching descriptions or source overlap produce proposed associations, never automatic acceptance. Reviewers must accept the association and supply its rationale. Every invariant needs an oracle. Missing architecture sections/capabilities, authentication context, boundary assumptions and unknowns become explicit obligations. Mapped context participates in source scope review even outside the selected prefix.

The bounded Python/SQLite ownership extractor can expose operations through offline discovery. For a uniquely matched subject and entry-point/caller flow, a local ownership guard can produce proposed support or contradiction. Evaluation inspects at most 20 distinct mapped entry-point Python files. Unsupported or ambiguous properties remain unknown. These observations do not establish caller authentication, deployment reachability or runtime behavior.

Review decisions are supported, contradicted, accepted assumption or unresolved. Invariants cannot be discharged as assumptions; resolved invariants require citations inside every mapped operation and explicit responses to structural proposals or supplied contradictory evidence. Reasoned local attestations satisfy accounting, not independent semantic acceptance. Decisions bind the exact plan/model and survive portable export as immutable partial submissions, merged hashes and explicit supersessions. A changed catalog/mapping requires a new review branch once either source or threat history has begun. Model-content revisions, arbitrary semantic reconciliation, document import and authenticated authorship remain future work.

Optional experiment links embed canonical-hashed replay receipts. Assessment checks the selected baseline/candidate source role and commit, mapped operation/case, file-backed citations, completed execution, verified cleanup, complete passing controls and agreement between assertions, outcome and disposition. Archive-only or inconclusive receipts cannot satisfy the link. These checks do not authenticate a verifier, certify an oracle or grant replay permission. Both source evidence acceptance and threat confirmation remain `not_performed`. A reviewed contradiction remains a problem even when adjudication is complete.

See the workflow, intake contracts, assessment implementation and regression tests.

9.3 Closed traceability graph

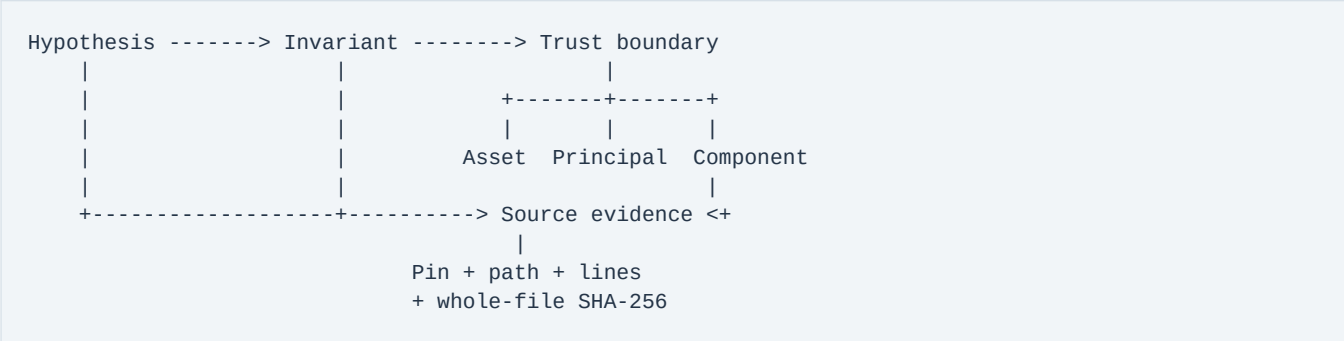


Figure 3. Required conceptual traceability. The aggregate validates references and scope; arrows do not imply observed runtime causality.

`ResearchReview` checks that linked records resolve and share tenant/project context. Every invariant remains linked even if it has no candidate hypothesis. Static observations are documented or inferred; they cannot claim observed runtime behavior merely because a line is cited. Missing transitions and unverified reachability remain explicit unknowns. [R15]

Scoped UUIDs are derived from tenant, project, commit, input digest, entity type, and local key. Evidence IDs also incorporate path/content within that snapshot. This supports stable identity for equivalent scoped input, while timestamps and other run metadata can vary. It does not supply cross-commit identity reconciliation or byte-for-byte deterministic report regeneration.

9.4 Current extraction algorithm

Template `python-sqlite-ownership-v3` uses Python ASTs and a bounded local-module resolver. At a high level it:

1. Selects pinned files from authorized upstream navigation locations.
2. Recognizes supported HTTP handlers that derive an identity from a request header.
3. Resolves direct local calls or explicit local imports into storage/service functions.
4. Tracks the identity argument by position through supported forwarding calls.
5. Examines constant SQLite query structure and positional bindings for ownership use.
6. Emits model entities, template-derived invariants, proposed hypotheses, per-hop citations, guard observations, and explicit gaps.

The resolver supports local calls, import aliases, relative imports, and explicit module aliases under the declared source prefix. Absolute-import resolution assumes that prefix is the module root. It does not execute package initialization, search installed packages, interpret arbitrary `sys.path` mutations, or recursively follow re-exports. A module path is limited to eight components. [R16]

Multi-hop analysis follows at most four callees after the HTTP handler, including storage, and shares a 128-step traversal budget across candidates in each selected file. Forwarders may return a call directly or assign a call result and return it, with an optional docstring. Arguments must be unchanged parameter names, and identity must occur exactly once. Renaming and positional reordering are supported.

Cycles, shadowed or ambiguous bindings, decorators, indirect calls, branching, transformations, extra statements, and unresolved authorization logic are not skipped optimistically. They result in explicit unsupported or unresolved observations. Every followed call, import declaration, and callee is cited against pinned bytes. [R16, R19]

9.5 Ownership and identity example

Consider a synthetic invoice handler that accepts an object identifier and an identity derived from `X-User`, forwards both to a service, and reads SQLite. The ownership template asks whether the storage lookup binds the requested object to the asserted identity. An unused identity parameter can support a proposed missing-owner hypothesis. A recognized matching owner predicate can remove that particular proposal in a fixed fixture.

Neither result proves that the identity itself is trustworthy. If the header can be supplied by the client and the upstream authentication context is unknown, the header-identity concern remains for both vulnerable and fixed variants. Thus the acceptance fixture yields two proposed hypotheses for the vulnerable form and only the shared header-identity concern for the recognized fixed form. This distinction prevents a local SQL guard from being mistaken for proof of end-to-end authorization. [R15, R16]

9.6 Prompt injection and interpretation limits

The deterministic compiler treats source and input text as data and never asks a model to interpret it during graph assembly. A small phrase-matching heuristic flags possible prompt-injection text for reviewers. It is neither comprehensive detection nor a defense claim for the upstream model's own agent loop. Source identity, structural matching, and output escaping reduce specific failure modes; they do not establish semantic correctness of every security assumption.

10. Ranking and bounded investigation planning

The research director ranks hypotheses with a transparent heuristic:

```
score = max(0, 100 * impact + round(20 * confidence)
               - 2 * hypothesis_assumption_count)

impact: low=1, medium=2, high=3, critical=4
```

Impact is the conditional severity of violating the linked invariant. Confidence is supplied in the input, not statistically calibrated by Sentinel. The assumption penalty uses the hypothesis input's assumptions; the final displayed hypothesis also includes boundary assumptions. Scores are scheduling priorities, not exploit probabilities, CVSS values, or confirmed finding severities. Ties are ordered by hypothesis key. [R15]

Claims on the same invariant with matching case-folded, whitespace-normalized descriptions are marked duplicate. This is syntactic duplicate handling, not semantic equivalence detection. Confidence below the default threshold of 0.2 is low-information. The first three eligible hypotheses are queued by default, with excess items deferred. Queue size can be configured within validated bounds.

The output includes proposed `Investigation` records and dispositions. No test, tool execution, privilege increase, or model spend is authorized merely by queue placement. The current director does not repeatedly investigate, learn from failed attempts, or execute autonomous stopping policies across attempts. Those are planned extensions.

11. Evidence integrity and auditability

11.1 Hash-chained local events

Policy and lifecycle events include actor, action, decision, resource, correlation identifier, time, and predecessor hash. A new event hash is computed from canonical JSON containing the previous hash and all event fields except its own hash:

```
H_i = SHA256(canonical_json(event_i without event_hash))
event_i.prev_hash = H_(i-1)
```

`verify_chain` recomputes hashes and checks predecessor continuity. The implementation redacts selected free-text fields before storage; it does not guarantee removal of every possible secret from every event or raw artifact. Local scan invocations use an in-memory audit store and export `audit.jsonl`. This export is useful for review, but it is not a crash-durable event journal during execution. [R17]

11.2 What the chain proves and does not prove

Given trusted retained events and an expected chain head, hashing helps detect modification, reordering, and missing interior records. A self-consistent chain alone does not authenticate its author. An attacker able to replace the whole export can recompute it, and removing a suffix is not detectable without an external expected head or record count. Sentinel does not currently sign or externally anchor audit heads.

PostgreSQL audit support exists, including tenant-scoped predecessor uniqueness to prevent forks under the implemented append path. Database-level restrictions against update/delete, migrations, and broader persistent entities remain backlog work. An application append-only interface is not equivalent to immutable storage. Local tenant IDs are organizational scope labels, not authenticated identities or proof of isolation between hostile tenants. [R3, R4, R17]

11.3 Reproducibility boundaries

Source pins, model IDs, upstream versions, arguments, and artifact hashes make a run inspectable and support controlled comparisons. They do not guarantee identical model output on rerun. The adapter configuration digest does not archive every internal upstream prompt or provider setting. Reproducible evidence means retaining enough context to investigate a claim and rerun an authorized experiment; generic synthetic replay is demonstrated in Section 14.6, while general independent verification remains future work.

12. Reports and operator experience

12.1 General scan report

`index.html` summarizes outcome, candidate findings, source omissions, run provenance, reported cost, and optional research. It preserves the distinction between complete and partial runs. An empty candidate list on a failed or interrupted run is not presented as a clean assessment. Even a completed run with no candidates does not establish absence of vulnerabilities. [R11, R18]

The research outcome is recorded separately as ready, unsupported, failed, or not run. A research parsing or coverage limitation does not rewrite the scanner's outcome. This permits honest reporting on repositories outside the current ownership-template scope.

12.2 Research review

`review.html` presents a searchable/filterable hypothesis queue, focused hypothesis details, source citations, assumptions, ranking explanations, architecture, and coverage gaps. It is self-contained, with bundled CSS/JavaScript and a restrictive content policy using asset hashes. Untrusted text and attributes are escaped. There are no remote report dependencies or telemetry calls. [R18]

Native links and disclosures, focus styles, responsive layouts, a no-JavaScript reading path, and print behavior are implemented. Printing includes all hypotheses and expanded details, regardless of active filters. Markup and JavaScript interaction tests exist, but full desktop/mobile visual and assistive-technology acceptance remains open. This paper does not claim an accessibility certification.

`sentinel research view` can validate saved review/extraction artifacts and create a new HTML view without rerunning repository analysis. It refuses overwrites and limits loaded saved input to 20 MB. A standalone view omits relative artifact-download links that would otherwise be misleading.

12.3 Assessment readiness and comprehensiveness

Local external-source experiments exposed an important distinction: a scanner can complete with zero findings while reading little substantive implementation. The report therefore separates execution, observed navigation, and assessment readiness. Telemetry states describe whether saved tool calls can be interpreted; they are not coverage scores. The report shows opened versus supplied files, requested ranges, failures and inventory-based path suggestions. Requested ranges may exceed EOF and do not prove returned content or comprehension. [R22]

Plan version 2 separates operation requirements from file scope accounting. A reviewer-supplied `OperationCatalog` embeds the existing research model of components, assets, principals, boundaries and invariants. Each invariant maps to a pinned operation range and explicit required context. Each operation receives five analysis requirements, while files receive scope dispositions. Multiple operations within one file remain distinct. Source/hash validation occurs before launch. Filename hints support navigation; without a catalog, operation discovery remains an explicit blocker. The plan digest binds its graph, inventory and requirements. [R22]

Snapshots now retain metadata for committed entries outside the selected prefix as well as selected text and omissions. Every entry needs a reasoned scope disposition. Reviewed context requires citations and operation links; irrelevant assets can be explicitly inapplicable. Unavailable surfaces remain blockers. Required context cannot be discharged by excluding its file, narrowing the prefix, or labelling it inapplicable. Outside-prefix source is not supplied to the model, although an authorized reviewer may inspect and cite it separately.

Reviewer evidence must contain trimmed analysis and a citation inside the operation's subject range. Checked source identity does not verify reasoning. Complete operation receipts are labelled `attested`, while `evidence_acceptance` remains `not_performed`. Reviewer identity is locally asserted. Boilerplate beyond the text threshold can still be poor analysis; the system does not mistake stronger syntax validation for semantic acceptance.

Assessment state	Meaning
unavailable	No assessment plan was prepared.
incomplete	Discovery, operation attestations, scope/context decisions, threat obligations, execution or unresolved work remains unmet.

Assessment state	Meaning
ready_for_review	Applicable source/threat receipts and scope decisions are complete; independent acceptance is still pending.

The offline `scan assess` command validates the authorized pin, reconstructed inventory, catalog and citations. Exports include immutable origin scan bytes, hash-bound attempt databases, a lineage manifest and up to 100 review revisions. Each continuation names the exact merged parent hash. Nonconflicting aspects accumulate; replacements require explicit supersession, and unresolved questions persist until explicitly resolved. Re-import recomputes navigation from original telemetry. Altered or missing bound artifacts fail loading. These hashes provide content integrity against the stored manifest, not signatures or authenticated authorship. Version-3 plans additionally bind threat mappings and a separate decision history. Unaccepted mappings and unresolved threat obligations block readiness even with complete source receipts. Reviewed contradictions and accepted assumptions stay visible; readiness does not imply safety. Research exports remain in the original directory. [R2, R22]

`scan run --attempts N` permits one to three passes with the existing upstream runtime; one is the default. Attempts share a policy engine, model allocation and wall-time ceiling. A queue names operation/context ranges, gaps, next actions and required evidence. The controller uses successful request intersections and new inventory-based recovery suggestions to measure navigation progress. Repeated or overlapping ranges add no progress. It retains every run, candidate and artifact, and stops on missing discovery, scope gaps, unavailable telemetry, unknown cost, exhausted budget, no progress, the attempt limit, or pending reviewer work. Covering requested ranges never supplies substantive attestations or runtime verification. Reported model costs are upstream estimates; cross-process budget reservations remain future work.

Synthetic catalogs exercise multiple operations, required context outside a selected prefix, and portable round-trip preservation. Real target catalogs and evidence remain local engagement inputs. Comprehensive assessment still needs broader automatic operation discovery, complementary methods, independent oracle acceptance and human adjudication. Detection gains require equal-budget comparisons and held-out cases distinct from development examples. [R22]

13. Operating a local assessment

13.1 Prerequisites and readiness

The local path requires Python 3.12 or later, uv, Git, Docker, the pinned scanner/proxy images, and configured OpenRouter credentials and model ID. PostgreSQL is not required for this workflow. The optional FastAPI service currently provides `/healthz` and database-dependent `/readyz`; it is not the scan execution interface. [R2, R20]

From the Sentinel checkout:

```
uv sync --frozen
cp .env.example .env
# Configure OPENROUTER_API_KEY and SENTINEL_REASONING_MODEL.
bash scripts/phase_a/install_0.sh
source scripts/phase_a/pins.env
docker pull "$EGRESS_PROXY_IMAGE"
uv run sentinel scan doctor
```

`scan doctor` checks local prerequisites and configuration without sending a model request. It does not prove account credit, model access, or future provider availability. The separate `openrouter smoke` command sends a small benign request; that client is a connectivity check, not the planned production ModelGateway. [R2, R20]

13.2 Scope and run

```
uv run sentinel scan init /absolute/path/to/repo \
  --owner "Authorized reviewer" \
  --out engagement.yaml --max-cost 1 --max-minutes 15

# Inspect the generated scope before running.
uv run sentinel scan run /absolute/path/to/repo \
  --manifest engagement.yaml --out var/assessment-001
```

Initialization creates a 24-hour manifest with the exact repository root and commit, source-read/static-analysis actions, a default 6,000 tool-call allowance, and one concurrent sandbox. The example sets a $1 model ceiling and 15-minute wall-time budget. Use a fresh output directory for each run; existing outputs are not overwritten. `--source-prefix` narrows the committed directory, and `--model`, `--max-cost`, and `--timeout` adjust permitted run configuration. [R2, R11]

The command emits periodic progress. Exit code 0 means the scanner completed and results were collected, irrespective of candidate count. Exit 1 indicates failure, denial, incomplete execution, or missing prerequisites. Operator cancellation exits 130. Upstream exit code 1, which can indicate high-severity claims, is translated as completed scanning rather than confused with the Sentinel CLI's failure code.

13.3 Artifact handoff

Artifact	Purpose
`index.html`	Offline candidate report and coverage summary.
`scan.json`	Run outcome, findings, snapshot, research, navigation and assessment status.
`threat-model-input.json`, `threat-model.json`	Preserved model, source checks, separate inferences and unresolved context.
`threat-mappings.json`, `threat-assessment.json`	Bound operation mappings, proposals, decisions and unresolved obligations.
`threat-review-template.json`, `threat-review.json`, `threat-review-history.json`	Editable handoff, merged decisions and immutable revision history bound by lineage.
`assessment-plan.json`	Pinned operation requirements, research graph and full scope inventory.
`assessment.json`	Attestation/scope progress, blockers and merged receipt digest.
`run.json`	Upstream/model identifiers, digests, lifecycle, cost, and output hashes.
`findings.json`	Canonical candidates with separate upstream metadata.
`source-snapshot.json`	Host-held source identity and omissions.
`artifacts.json`	Collected evidence/log artifact identities and hashes, when available.
`audit.jsonl`	Local policy and lifecycle audit export.
`attempts.json`, `controller.json`	Retained attempt records, pending work and stop reason.
`origin-scan.json`, `lineage.json`, `review-history.json`	Portable assessment origin, integrity bindings and review revisions.
`upstream/`	Per-attempt reports, logs, databases and supplied snapshots.
`research/`	Extraction receipt, structured input/review, Markdown, and HTML when produced.

A reviewer should first establish whether the scan completed, then inspect omissions and candidate evidence, then examine research assumptions and proposed tests. Raw upstream files are evidence inputs, not trusted executable reports. Local artifacts can contain source and sensitive context; retention and sharing remain operator responsibilities in this version.

14. Validation and experimental evidence

14.1 Engineering validation

The original implementation checkpoint recorded 266 passing Python tests, four PostgreSQL-dependent skips in the local environment, and six passing JavaScript interaction tests. Ruff lint/format checks and strict mypy checks passed. CI configures PostgreSQL and runs Python tests with coverage plus the JavaScript suite. These are engineering-validation results, not vulnerability-discovery benchmark scores. [R19, R21]

The tests cover policy denial and expiry, revocation, tenant consistency, proof-state guards, schema drift, stale citations, source pinning, path/symlink restrictions, report tampering, cancellation/cleanup behavior, and partial outcomes. Extraction controls include vulnerable/fixed fixtures, cross-file imports, multi-hop forwarding, cycles, depth/step limits, ambiguous bindings, reordered parameters, and unsupported service logic. Report tests cover escaping, content-policy asset hashes, saved-artifact handling, and interaction logic.

The November 14 assessment extension passed 308 Python tests with four PostgreSQL skips, six JavaScript tests, lint/format checks and strict typing. New controls cover blank input, operation-range citations, scope context, portable lineage, partial-receipt merging, conflict handling, bounded follow-ups and shared attempt budgets. These checks validate accounting and acceptance behavior, not reviewer semantic correctness or detection gains.

The generic external-case replay extension passed 342 Python tests with four PostgreSQL skips, all six JavaScript tests, lint/format and strict typing checks. New tests reject failed controls, duplicate or malformed probes, abnormal exits and unverified cleanup; browser exports escape untrusted evidence.

The explicit threat-model extension passed 366 Python tests with four PostgreSQL skips and all six JavaScript tests, plus lint, formatting and strict typing. Its synthetic controls cover uncited assumptions, wrong commits, stale references, reported contradictions, attachment tampering and overrides, unchanged authorization, provisional extraction, bounded guidance, HTML escaping and portable preservation. An offline check of the included vulnerable fixture retained a reviewer-reported ownership contradiction with a checked source reference and eight unresolved context/review gaps. No paid model call or target execution was needed for that check; it did not automatically establish the truth of the interpretation.

The November 15 threat-driven assessment extension passed 404 Python tests with four PostgreSQL skips, six JavaScript tests, lint/format and strict typing checks. Controls cover proposed versus accepted mappings, stale bindings/citations, unresolved assumptions, excluded mapped context, review supersession/tampering, portable round trips, escaped review text and invalid experiment links. A live network-disabled replay of owned vulnerable/fixed storage helpers reproduced baseline cross-owner disclosure and passed candidate expectations, with both control modes passing and cleanup verified. Synthetic review records linked each result to a mapped invariant and exercised readiness; this was not a paid upstream scan, authenticated human adjudication or a detection benchmark. The helper probe does not establish HTTP reachability or caller identity.

The evaluation extension passed 420 Python tests with four PostgreSQL-dependent skips, plus six JavaScript tests, lint/format and strict typing. Controls cover stale collections, duplicate assignments, unsupported/unknown coverage, incomplete runs, missing costs, legacy inventories, held-out timing/code changes, source citations, false-positive accounting and escaped reports. These tests validate accounting and workflow integrity rather than the truth of every judgment.

Many adapter tests use fake processes and runners to exercise state transitions deterministically. They are complemented by recorded live Docker/provider runs, but they do not prove absence of containment bypasses on

all hosts or all upstream versions. Database tests skipped locally must not be reported as passing in that environment.

14.2 Framework bake-off

ADR-001 records four 0.security runs over public synthetic vulnerable/safe-control fixtures. Each configuration ran once; the fixtures were not held out. The following table preserves the reported observations. [R5]

Fixture / model	Outcome and reported cost	Interpretation
Vulnerable / sonnet-4.6	Cost stop; 188 s; about $1.55; 3 claims	Reported all three seeded classes; incomplete investigation.
Safe control / sonnet-4.6	226 s; about $1.22; 1 claim	Header-identity issue outside seeded classes.
Vulnerable / gpt-4o-mini	94 s; about $0.03; 1 claim	Reported one of three seeded classes.
Safe control / gpt-4o-mini	135 s; about $0.11; 11 claims	Seven claims classified as seeded-class false positives; four require further adjudication.

The model identifiers were `anthropic/claude-sonnet-4.6` and `openai/gpt-4o-mini`. In the safe-control mini run, two SQL-injection claims against the ground-truth-safe query were labeled `verified` upstream. That observation directly motivates Sentinel's refusal to inherit an upstream verification label. The sonnet vulnerable run reported approximately $1.548 against a $1.50 ceiling, demonstrating the cost-accounting qualification described earlier.

Two PentAGI runs did not produce final assessments within their limits: the vulnerable fixture stopped after about 1,087 seconds and $3.20 against a $3 watchdog cap; the safe control reached about 1,811 seconds and $2.71 without a final report. These observations support the present integration choice, but the small, unequal set of runs cannot establish a general ranking of frameworks. Licensing, dependency/image review, repeated measurements, public API/MCP evaluation, held-out fixtures, and human sign-off remain ADR open items. [R5]

14.3 End-to-end local CLI validation

Decision D-016 records a completed scan of the included vulnerable fixture using `openai/gpt-4o-mini`:

- Target commit: `7d66335e3db73e79743838007672bc3ec054e043`.
- Run identifier: `284007dc-6029-4bad-9e2c-6f8851f34f37`.
- Output: one candidate finding and two proposed research hypotheses.
- Upstream-reported cost: `$0.02332125`.
- Audit-chain verification and Docker container/network teardown were checked.
- Local evidence directory: `var/scan-cli-validation-2/`, intentionally excluded from Git.

This is an integration demonstration: authorization, snapshot preparation, provider-connected scanning, normalization, research enrichment, reports, audit, and teardown operated together. It is not evidence of external-repository accuracy. The raw local evidence is not distributed with this paper; the committed decision log is the retained narrative record. The first operator-selected external repository scan and human triage remain open. [R3, R4]

14.4 Lessons from external examples

Local external-source experiments showed that a completed scan with zero candidates can still have sparse source reads, repeated navigation failures and unsupported semantic extraction. These observations motivated independent execution/navigation/assessment states, explicit operation catalogs and bounded follow-up. They are design inputs, not committed benchmark data or claims about a particular repository. Target-specific source, harnesses and results remain in ignored local case studies. Committed tests use synthetic sparse-navigation records. [R23]

14.5 Evaluation still required

A credible product scorecard requires pinned vulnerable/fixed pairs, genuinely held-out targets, explicit ground truth where possible, repeated configurations, and human adjudication. Precision should count adjudicated valid claims over reviewed claims; recall requires a defined expected set. Cost per confirmed finding is meaningful only after independent confirmation exists. Failure and partial-run rates, reviewer effort, coverage omissions, reproducibility, and policy violations should accompany headline detection metrics.

The build specification proposes targets for recall, precision, chain recall, reproducibility, and report latency. They are engineering aspirations, not measured Sentinel performance. No claim of zero-day discovery, calibrated confidence, or general security assurance follows from the present fixtures. [R1]

14.6 Generic behavioral replay and fixture browser validation

`replay run --spec` accepts operator-supplied case packs. The spec declares source roles and full pins, selected files or archives, sandbox harness files, operation IDs, scenarios, expected JSON fields, negative-control fields and limitations. A policy-gated Docker runner disables networking, uses an unprivileged UID, read-only source/root mounts, bounded resources and checked teardown. Case code runs only in the container. Spec, source, harness and artifact hashes bind observations; JSON, Markdown and offline HTML expose outcomes, controls and coverage limits. [R24]

The host compares observations to predeclared expectations; it does not load target-specific oracle plugins. Every baseline/candidate scenario needs a target probe, positive control and negative control. Positive controls must match all declared fields. Negative controls must change the designated field while preserving the others. Missing, duplicate, unknown, malformed, non-finite, wrong-type, crashed or interrupted observations and unverified cleanup prevent a behavioral conclusion. Oracle correctness remains an operator/reviewer responsibility. Replay never automatically promotes findings or clears semantic assessment gaps.

A live synthetic calculator replay exercised all six variant/mode combinations. It reproduced the intentionally incorrect baseline sum, passed the candidate and both controls, verified cleanup and emitted browser evidence. This demonstrates transport and comparison behavior, not vulnerability detection. The earlier included web fixture scan also completed with two candidates and two proposed hypotheses; browser review retained unverified status and explicit scope/discovery gaps. [R24]

Real repositories, dependency versions, build scripts, domain-specific oracles and outcome reports stay in ignored `var/case-studies/` or outside the product repository. Further examples use the same contracts and inform synthetic regressions. Version-2 replay receipts replace the original experiment-bound shape with generic case-pack/profile and spec identity. Old receipts remain historical artifacts and are not silently upgraded.

14.7 Frozen protocols and cross-repository evaluation

The local evaluation workflow declares repository pins, source selections, requirements, operation catalogs, expected properties, variants and fixture/development/held-out splits. Freeze validates source identities and scope, records manifests and a Sentinel Python-source fingerprint, and exports scanner inputs separately from expected labels. Requirements guide the scan, so this is threat-guided evaluation rather than blind discovery. Protocol/reference labels remain operator assertions and need oracle validation. [R25]

Scoring consumes existing scan bundles without model calls or target execution. It rechecks origins, telemetry/history bindings and the authorized source inventory, then recomputes assessment. Historical imports remain retrospective; missing outside-prefix inventory can be reconstructed without rewriting old scan bytes. Held-out eligibility requires unchanged implementation, matching pre-scan context/manifest digests and attempts after freeze. These are local integrity checks, not authenticated timestamps or proof of novelty to a model's training data.

The scorecard distinguishes selected/omitted/excluded source, observed reads, declared operations, automatically extracted invariants, reviewed operations/invariants, open context and untested invariants. Counts describe the declared catalog, not every repository operation. Unsupported extraction remains visible even when the scanner opens all selected files.

Evaluation decisions bind the exact collected evidence, identify an agent or claimed human reviewer, and cite the relevant source operation. Detected, missed, clean, false-alarm and inconclusive outcomes remain distinct. Completed empty scans require explicit adjudication before counting as misses or clean controls. Unexpected findings are not automatically false positives. Duplicate claims do not inflate detections. No evaluation decision changes finding proof states or supplies independent semantic acceptance.

Metrics separate splits and preserve unknown values. Recall uses all declared violated properties; false-alarm rate uses all declared negative properties. Precision uses distinct detected properties and false-positive candidate claims, excluding duplicate detections. Missing/incomplete scans, unadjudicated candidates, unresolved relevant denominators or invalid held-out protocols suppress rates. Costs sum reported attempt costs only when all are known; elapsed time sums attempt durations. Configuration differences do not establish causal improvement or equal effective reasoning effort.

The first local pilot exercised an owned fixture, a retrospective external development repository and a materially different external vulnerable/fixed pair held out from implementation tuning. Controlled replay established the narrow variant distinction and passed positive/negative controls. Agent adjudication and external evidence remain local; human acceptance, repeated matched-budget runs, broader discovery and independent evaluation remain open. External target implementations, labels and results are not committed product fixtures. [R25]

15. Assurance limits and residual risks

The current system is designed to expose uncertainty rather than conceal it. The following qualifications are material to interpreting a scan:

Area	Current boundary	Consequence
Model reasoning	Upstream output remains untrusted	Hallucinated findings and false negatives remain possible.
Research coverage	Narrow AST/template recognition	Unsupported structure cannot be treated as a negative result.

Area	Current boundary	Consequence
Threat context	Bounded proposals and local reviewer attestations	Accepted assumptions and adjudicated contradictions do not establish safety.
Verification	Schema guard and external case-pack replay	Expected values require review; no automatic independent confirmation is provided.
Provider handling	Host allowlist, tunneled TLS	Source processing and retention are not enforced by the proxy.
Budgets	Per-process admission and upstream cost reports	No hard shared quota or exact real-time billing enforcement.
Host/runtime trust	Docker daemon, local image, operator filesystem	Compromise of these components can undermine containment and evidence.
Artifact integrity	Hashes and local chain without external signing	Whole-history replacement or suffix truncation needs an external trust anchor to detect.
Persistence	Local files and in-memory scan audit	Abrupt host/process failure can lose not-yet-exported state.
Supply chain	Pinned source, tagged scanner image	Dependency/image review and stronger image attestation remain open.
Multi-tenancy	Scoped records, no authenticated service	Tenant IDs must not be treated as hostile-tenant isolation.

Further operational gaps include comprehensive output quotas, general secret minimization, retention controls, authenticated approvals, durable cancellation recovery, and full UI acceptance. These are reasons to scope the current product as a local pilot workflow, not to infer that the implemented safeguards have no value. [R2-R7]

16. Planned platform extensions

16.1 Finish Phase C against real repositories

The next research work is broader architecture extraction across frameworks and languages, richer service logic and caller discovery, additional invariant templates, incremental graph refresh, and persistent stopping conditions across attempts. A reviewer must be able to distinguish source observation from inferred security intent and trace each hypothesis to an asset, boundary, invariant, and evidence requirement. That traceability is implemented for the current slice; complete Phase C acceptance remains open. [R4, R15]

16.2 Typed causal reasoning

The planned chain engine treats weaknesses as transitions with a principal, preconditions, required state, action class, postconditions, evidence, and confidence. A candidate edge requires compatible identity, state, environment, and capability assumptions, with predecessor postconditions satisfying successor preconditions. A graph path is only a proposal until its critical links are independently validated.

The build specification suggests an initial graph-search implementation and a heuristic incorporating impact, reachability, evidence strength, and feasibility. Neither that chain-ranking formula nor typed matching is

implemented by the present research director. Its current ranking formula is the hypothesis heuristic in Section 10. Keeping these two concepts separate prevents a linked source trace from being represented as a validated exploit chain. [R1, R8]

16.3 Independent safe verification

The planned verification path provisions disposable environments, synthetic identities, fixtures, approved tests, and explicit assertion oracles. It records `reproduced`, `not_reproduced`, `skipped`, or `error`, then requires an independent reviewer/verifier decision. Vulnerable/fixed pairs and negative controls should challenge both the claimed weakness and its remediation.

A useful acceptance gate is a synthetic multi-step case reproduced within policy, with incompatible-principal and impossible-state controls rejected with reasons. The existing finding schema provides a necessary representation constraint; broader runner integration, independent evidence acceptance, and the workflow are still required. [R1, R4]

16.4 Hosted operations and domain specialization

Before authenticated multi-user operation, Sentinel needs migrations, persistent project/engagement/finding/evidence records, tenant-aware authentication and authorization, idempotent APIs, shared budget reservations, durable job state, and database-enforced audit restrictions. A production ModelGateway must add structured-output handling, bounded repair, retries/backoff, provider policies, and per-task accounting. The current connectivity client supplies none of that full workflow. [R4, R20]

Financial/protocol and AI-agent invariant packs are proposed domain extensions. Continuous regression checks, CI/PR integration, and an optional PentAGI adapter are contingent roadmap items rather than delivered features. The intended progression is driven by held-out evidence and measured coverage gaps, not by the presence of empty architectural modules.

17. Engineering tradeoffs

Sentinel's present architecture favors inspectability and a narrow operational surface. A local CLI and static reports allow useful scans before a hosted control plane exists. The cost is limited collaboration, persistence, and centralized policy administration. CLI integration with a pinned upstream avoids unstable internal API coupling, while requiring explicit translation of statuses and stronger compatibility checks on upgrades.

Deterministic AST templates provide explainable source traces and useful negative controls without additional model calls. Their cost is restricted coverage: recognizing a forwarding function is far easier than proving arbitrary authorization logic. Explicit gaps are therefore part of the output contract rather than exceptional failures to hide.

Separating candidate reports from research enrichment makes the system useful on more repositories today. Separating hypotheses from verification preserves the ability to add stronger evidence later without changing the meaning of existing results. These tradeoffs define Sentinel's current contribution: a constrained and inspectable bridge from authorized source review to falsifiable security research, with broader assurance dependent on the unfinished verification and evaluation work.

Appendix A. Module and source map

Responsibility	Repository location
Canonical records and enums	`src/sentinel/core/`
Manifest decisions and quotas	`src/sentinel/policy/`
Upstream lifecycle and egress	`src/sentinel/adapters/`
Snapshot, extraction, graph, ranking	`src/sentinel/research/`
Local orchestration and candidate report	`src/sentinel/scanning.py`, `src/sentinel/scan_report.py`
Assessment planning, readiness and navigation observations	`src/sentinel/assessment.py`, `src/sentinel/assessment_store.py`, `src/sentinel/scan_controller.py`, `src/sentinel/scan_coverage.py`
Threat context, mappings, decisions and browser presentation	`src/sentinel/threat_model.py`, `src/sentinel/threat_assessment.py`, `src/sentinel/threat_workflow.py`, `src/sentinel/threat_report.py`
Frozen evaluation and scorecards	`src/sentinel/evaluation/`, `src/sentinel/cli/evaluation.py`
Generic behavioral replay and host comparison	`src/sentinel/replay/`
CLI commands	`src/sentinel/cli/`
Audit and redaction	`src/sentinel/evidence/`
Database/API skeleton	`src/sentinel/db/`, `src/sentinel/api/`
Connectivity smoke client	`src/sentinel/gateway/`
Cross-language schemas	`contracts/v1/`
Controls and regression checks	`tests/`, `tests/js/`

Appendix B. Document provenance and regeneration

The original implementation baseline is `86fc22409a3e897c2976a7f19ccec496afeae031`; assessment work includes `2a16134e84a3a1200a84dc1d0093951fcb1b8af7` and the later replay and threat-driven extensions in Sections 9.2, 12.3 and 14. This implementation description is not an independent audit or certification; later changes can invalidate its conclusions.

Markdown is the source of truth. Rendering requires Pandoc and Python 3 with ReportLab; Poppler supports visual inspection. These are document tools, not runtime dependencies.

```
python3 scripts/render_whitepaper.py
```

References

All references below are repository primary sources. Relative links are resolved from this document's directory; the PDF preserves relative links to the same local repository files for this revision. Local ignored run artifacts are identified in Section 14.3 but are not represented as publicly available evidence.

[R1] YeleTech Sentinel build specification. Intended product, architecture, verification, evaluation, and phase gates.

[R2] Local source scan guide and README. Current commands, limits, exits, and operating assumptions.

[R3] Decisions and deviations, D-001 through D-025. Implementation choices and recorded live-run observations.

[R4] Backlog. Completed slices, open acceptance work, and infrastructure gaps.

[R5] ADR-001: runtime selection and bake-off data. Historical fixture results, interpretation, and pending sign-off.

[R6] 0.security adapter. Launch, limits, lifecycle, normalization, and report handling.

[R7] Adapter contract and source-context projection. Capabilities, run records, candidates, and navigation.

[R8] Domain records, base envelope, and enums. Canonical semantics and confirmation guard.

[R9] Policy engine and manifest. Authorization, approvals, budgets, and scope.

[R10] Pinned source and scan snapshot. Git reads, citations, materialization, omissions, and limits.

[R11] Scan orchestration and scan CLI. Operator workflow and separate enrichment status.

[R12] Egress sandbox. Proxy policy, network setup, denied-host extraction, and teardown.

[R13] Schema exporter and versioned schemas. Cross-language interface and schema inventory.

[R14] Upstream pins and scanner installation. Reused runtime versions and build inputs.

[R15] Phase C guide, research models, and director. Input, graph closure, identities, ranking, and planning.

[R16] Extraction, ownership analysis, and module resolution. Supported patterns and bounded traversal.

[R17] Audit log, redaction, and database models. Hashing and persistence boundaries.

[R18] Candidate report, research HTML, and review assets. Offline presentation and interaction.

[R19] Python tests and JavaScript report tests. Contract, policy, extraction, lifecycle, and report controls.

[R20] API skeleton, OpenRouter smoke client, and project metadata. Implemented platform surface and dependencies.

[R21] CI workflow. Automated lint, type, Python/database, and JavaScript checks.

[R22] Assessment planner and gate, coverage parser, and assessment controls. Operation attestations, scope decisions and source validation; revision store, bounded controller, and synthetic assessment tests.

[R23] Assessment design review and synthetic navigation fixture. Generalized lessons and regression coverage.

[R24] Case-pack replay guide, synthetic replay fixture, replay controls and implementation. Generic execution, expected-value comparison and limitations.

[R25] Evaluation workflow, contracts and scoring and regression tests. Protocol freezing, explicit denominators, coverage and local adjudication.