Symbia Seed Audit Bundle — Living Specification

This document accompanies the Symbia Seed audit bundle generated from a live runtime (e.g. Replit-hosted Seed). It is written for engineers, auditors, security reviewers, and technically literate stakeholders who want to independently evaluate what the bundle proves, how to read it, and how to verify its claims without privileged access.

This is a single source of truth: it consolidates explanation, walkthroughs, checklists, and compliance framing in one place.

1. What This Bundle Is (Precise Definition)

This bundle is execution-layer evidence.

It is the complete, externally inspectable output of two canonical Symbia Seed audit suites executed against a live runtime:

- Visibility
- Enforcement

Together, they assert a narrow but critical claim:

This runtime enforced specific execution-layer invariants at runtime, and emitted sufficient evidence for those claims to be independently verified after the fact.

The bundle is not logs, not screenshots, not model evals, and not a demo artifact. It is a forensic record.

2. What This Bundle Is Explicitly Not

To avoid category errors, this bundle does not attempt to prove:

- model reasoning quality
- factual correctness of outputs
- usefulness or UX quality
- alignment in general
- safety in all environments

A PASS result does not mean “the system is good.”

It means only:

Specific execution-layer guarantees held, and the runtime did not cheat.

Anything outside that scope is intentionally excluded.

3. High-Level Structure of the Bundle

You should expect the bundle to contain:

- Per-audit `` files (authoritative)
- Per-audit `` files (CI-compatible, derivative)
- Artifact references (paths, hashes, signatures)
- Trace and refusal artifacts produced during execution
- An audit index tying all outputs together

Important: the filesystem layout is part of the contract. The audits are discoverable and interpretable without introspecting runtime internals.

4. The Two Audit Suites

4.1 Visibility Suite — “Can This System Be Examined?”

Visibility audits assert that execution leaves evidence.

Each audit:

- runs against a live Seed process
- executes a real action (boot, FS access, network call, restart, trace emission)
- collects only externally emitted artifacts
- asserts over those artifacts — not internal state

Visibility answers questions like:

- Can identity be observed without inspecting secrets?
- Are allowed and denied surfaces externally visible?
- Can continuity be demonstrated across restarts?
- Can traces be followed without ambiguity?

If a property cannot be proven from emitted artifacts, it does not count as visible.

Visibility does not test whether something is allowed or forbidden — only that allowance or denial is externally attestable.

4.2 Enforcement Suite — “Can This System Cheat?”

Enforcement audits assert that certain violations cannot occur silently.

These audits deliberately attempt to violate execution-layer rules, then verify that:

1. the violation is blocked
2. the block produces a first-class artifact
3. the artifact is attributable, signed, and trace-linked

Examples of enforced guarantees:

- actions without actor identity are rejected
- state cannot mutate without a signed diff
- silent refusal is forbidden
- tampered diffs are detected
- refusals must be materialized as artifacts

Enforcement depends on Visibility. Blocking without evidence is not enforcement.

5. How to Read the Output Files

5.1 summary.json (Authoritative)

This is the canonical result for each audit.

Typical fields include:

- audit_id
- suite (visibility | enforcement)
- start_time / end_time
- result (PASS | FAIL)
- per-assertion results
- artifact references (paths, hashes, signatures)

If summary.json and junit.xml disagree, `` wins.

5.2 junit.xml (Derivative)

This file exists solely for CI and ecosystem compatibility.

It:

- mirrors pass/fail outcomes
- may omit context
- is never authoritative

Do not rely on it alone.

6. Interpreting PASS vs FAIL

6.1 What a PASS Means

A PASS means:

- the action occurred
- required evidence was emitted
- all assertions over that evidence held

A PASS does not mean:

- the system is safe in general
- the model behaved correctly
- the environment was non-adversarial

PASS is intentionally narrow and binary.

6.2 What a FAIL Means

A FAIL always indicates one of three execution-layer failures:

1. something was allowed that should not have been
2. required evidence was not emitted
3. emitted artifacts were ambiguous or unverifiable

There is no expected-fail mode.

Any FAIL makes the runtime non-conformant.

7. Step-by-Step: Walking a Single Audit

Use this process for any audit in the bundle:

1. Locate `` for the audit ID
2. Confirm result: PASS
3. Inspect listed artifacts (paths + hashes)
4. Verify:
5. artifact exists
6. hash matches
7. signature/identity metadata is present
8. Cross-check timestamps against start_time / end_time
9. (Optional) Compare to junit.xml for CI parity

No debugger. No privileged access. No internal state.

If you cannot verify a claim from emitted artifacts, the audit does not count.

8. Independent Verification Checklist

This checklist is the normative, fail-closed procedure an independent verifier must follow after obtaining a bundle via the canonical manual procedure in Section 9.3.

All steps are mandatory. If any step cannot be completed using only the bundle contents and standard tooling, the run is non-conformant.

8.1 Bundle Integrity

-

8.2 Audit Index Validation

-

The audit index is the authoritative manifest for the run.

8.3 Per-Audit Result Validation

For each audit listed in the index:

-

If any audit reports FAIL, verification stops and the run is non-conformant.

8.4 Artifact Existence and Hash Verification

For each artifact referenced in summary.json:

-

Missing or hash-mismatched artifacts invalidate the audit.

8.5 Identity and Attribution Checks

-

8.6 Refusal and Denial Evidence

For audits asserting denial or refusal behavior:

-

Silent failure or dropped requests are non-conformant.

8.7 State Mutation and Diff Integrity

For audits involving state changes:

-

Any unsigned or unverifiable state mutation invalidates the run.

8.8 Replay and Continuity Assertions

Where replay or continuity is claimed:

-

8.9 Cross-Suite Consistency

-

Visibility without Enforcement, or Enforcement without Visibility, is insufficient.

8.10 Final Determination

-

If and only if every checkbox is satisfied, the execution-layer guarantees hold.

Any unchecked item results in a non-conformant determination.

9. Third-Party Auditor Instructions (Normative)

This section defines how an external, independent auditor should evaluate a Symbia Seed audit bundle. These instructions are normative and are written to minimize reliance on trust, tooling assumptions, or insider knowledge.

9.1 Auditor Preconditions

An auditor must not:

- run custom instrumentation inside the Seed runtime
- attach a debugger or profiler
- inspect private memory or secrets
- rely on verbal explanations from operators

An auditor may:

- inspect the bundle contents

- recompute hashes
- verify signatures
- replay documented verification steps
- use standard OS tooling (shell, hash utilities, XML/JSON viewers)

The auditor is assumed to be hostile-but-fair.

9.2 Required Inputs

The auditor should be provided with:

1. the complete audit bundle (unmodified)
2. this Living Specification
3. the Seed version identifier associated with the run

No additional context is required.

9.3 Manual Audit Execution Procedure (Reference)

The following manual steps describe how an auditor or reviewer may independently execute the audits against the hosted Symbia Seed control plane. These steps are reference-grade: they are not required to validate an already-downloaded bundle, but they define the canonical way a bundle is produced.

These commands assume a POSIX shell, curl, and jq.

Step 0 — Health Check (Mother Runtime)

```
curl -s https://console.symbia-labs.com/api/health | jq
```

Confirms that the supervising (mother) runtime is alive and reachable.

Step 1 — Spawn a Fresh Child Runtime

```
spawn=$(curl -s -X POST https://console.symbia-labs.com/api/spawn) echo "$spawn" | jq
SID=$(echo "$spawn" | python -c "import sys,json; print(json.load(sys.stdin)['session_id'])")
```

This creates an isolated child Seed instance. All subsequent audit artifacts are scoped to this session ID.

Step 2 — Verify Workers in the Child

```
curl -s https://console.symbia-labs.com/api/c/$SID/workers/status | jq
```

Confirms that required workers are present and running before audits are executed.

Step 3 — Run Visibility Audit Suite

```
curl -s -X POST https://console.symbia-labs.com/api/c/$SID/tests/run \ -H "Content-Type:
application/json" \ -d '{"suite":"visibility"}' | jq
```

Executes the full Visibility suite against the live child runtime.

Step 4 — Run Enforcement Audit Suite

curl -s -X POST https://console.symbia-labs.com/api/c/$SID/tests/run \ -H "Content-Type: application/json" \ -d '{"suite":"enforcement"}' | jq

Executes the full Enforcement suite. These audits deliberately attempt to violate execution-layer rules.

Step 5 — Inspect the Audit Index

curl -s "https://console.symbia-labs.com/api/c/$SID/logs/file?path=audit/index.md"

The audit index is the authoritative manifest for the run. It lists all audits, results, and artifact locations.

Step 6 — List Available Logs and Reports

curl -s "https://console.symbia-labs.com/api/c/$SID/logs/index" | jq

Enumerates all files emitted by the runtime for the session.

Step 7 — Fetch Specific Audit Artifacts (Examples)

curl -s "https://console.symbia-labs.com/api/c/$SID/logs/file?path=visibility/vis_boot_identity/summary.json" | jq curl -s "https://console.symbia-labs.com/api/c/$SID/logs/file?path=enforcement/enf_diff_factory_accept_valid/summary.json" | jq

Demonstrates direct retrieval of per-audit summaries without privileged access.

Step 8 — Download the Full Session Bundle

curl -s "https://console.symbia-labs.com/api/c/$SID/logs/bundle" -o symbia-logs-$SID.zip

This produces the exact bundle evaluated by third-party auditors.

9.4 Verification of a Downloaded Bundle

When validating an existing bundle, the auditor must not re-run the system. Only the emitted artifacts are considered.

Verification follows the procedure in Section 8.

9.5 Handling FAIL Results

If any audit fails verification:

- the auditor must mark the entire run non-conformant
- partial credit is not permitted
- downstream claims relying on the run must be suspended

There is no concept of "expected failure" or "acceptable deviation".

9.6 Auditor Output Expectations

A third-party auditor should produce a short report containing:

- Seed version evaluated
- spec version used
- audit IDs reviewed
- verification outcome (conformant / non-conformant)
- list of any missing or ambiguous artifacts

The report need not restate audit logic. It must state only whether evidence sufficed.

9.7 Scope Boundary

The auditor is not responsible for evaluating:

- model intelligence or alignment
- user experience quality
- prompt correctness
- downstream application behavior

Attempting to do so invalidates the audit.

9.2 Required Inputs

The auditor should be provided with:

1. the complete audit bundle (unmodified)
2. this Living Specification
3. the Seed version identifier associated with the run

No additional context is required.

9.3 Verification Procedure

For each audit in the bundle, the auditor should:

1. Locate the corresponding summary.json
2. Confirm the audit_id and suite
3. Verify result == PASS
4. Enumerate all referenced artifacts
5. Confirm each artifact:
6. exists at the stated path
7. matches the recorded hash
8. contains required metadata (identity, timestamps, linkage)
9. Confirm timestamps fall within the audit execution window

If any referenced artifact is missing, ambiguous, or unverifiable, the audit must be treated as FAIL, regardless of reported status.

9.4 Refusal and Denial Audits

For audits involving refusal or denial:

- the auditor must locate a material refusal artifact

- the artifact must be first-class (not implicit error text)
- the artifact must be trace-linked and attributable

Silent failure, dropped requests, or out-of-band error paths are non-conformant.

9.5 Replay and Continuity Audits

Where replay or continuity is asserted:

- the auditor must confirm that replay sensitivity is explicitly documented
- identical inputs under differing execution conditions must be distinguishable

Replay is a consistency check, not a simulation. Narrative reconstruction is insufficient.

9.6 Handling FAIL Results

If any audit fails verification:

- the auditor must mark the entire run non-conformant
- partial credit is not permitted
- downstream claims relying on the run must be suspended

There is no concept of "expected failure" or "acceptable deviation".

9.7 Auditor Output Expectations

A third-party auditor should produce a short report containing:

- Seed version evaluated
- spec version used
- audit IDs reviewed
- verification outcome (conformant / non-conformant)
- list of any missing or ambiguous artifacts

The report need not restate audit logic. It must state only whether evidence sufficed.

9.8 Scope Boundary

The auditor is not responsible for evaluating:

- model intelligence or alignment
- user experience quality
- prompt correctness
- downstream application behavior

Attempting to do so invalidates the audit.

9. Why Visibility and Enforcement Are Paired

The suites are separate because they fail differently:

- Visibility without Enforcement → observable but untrustworthy
- Enforcement without Visibility → correct but unverifiable

Symbia Seed requires both:

- Visibility PASS ⇒ the system can be examined
- Enforcement PASS ⇒ the system cannot cheat

Anything else collapses into performative compliance.

10. Compliance & Regulatory Framing (Optional)

This bundle maps cleanly to common institutional requirements:

- Auditability — reconstructable execution
- Attribution — actor-bound actions
- Non-repudiation — signed artifacts
- Change control — diff-based state mutation
- Default deny — capability-based access
- Explainability (operational) — why an action was blocked

Important: this is execution governance, not model explainability.

11. Common Misinterpretations to Avoid

- “It passed audits so it’s safe” X
- “This evaluates the model” X
- “These are just logs” X
- “Replay is best-effort” X

Replay is a consistency check, not a narrative reconstruction.

12. Summary (One-Paragraph Version)

This audit bundle is a cryptographically attributable, externally verifiable record showing that a live Symbia Seed runtime enforced specific execution-layer invariants and emitted sufficient evidence to prove it. It does not judge intelligence, quality, or safety in general. It establishes eligibility for trust, not confidence.

13. Living Specification Status

This document is a living specification.

It is expected to evolve alongside Symbia Seed releases and audit suite revisions. Changes to this document are governed by the same execution-layer philosophy it describes.

13.1 Versioning Model

- This document is versioned independently from code, but references concrete Seed versions where relevant.
- Breaking changes to audit semantics must be reflected here before or at the time of release.
- Clarifications and explanatory improvements may be added without changing audit behavior.

Recommended version tag format:

AuditSpec vX.Y — Compatible with Seed >= A.B.C

13.2 What Qualifies as a Spec Change

A change to this document is required if any of the following occur:

- a new audit is added
- an audit assertion is strengthened or weakened
- artifact shapes or required fields change
- replay or verification semantics change
- PASS / FAIL interpretation changes

Pure refactors, runner changes, or internal implementation details do not require spec updates unless they alter externally visible evidence.

13.3 Canonical vs Descriptive Sections

This document contains two kinds of content:

Canonical (normative)

- definitions of PASS / FAIL
- required artifacts
- verification rules
- suite pairing requirements (Visibility + Enforcement)

Descriptive (explanatory)

- walkthroughs
- examples
- compliance framing
- narrative guidance

In case of conflict, canonical sections win.

13.4 Backward Compatibility

Older audit bundles remain valid evidence under the spec version that was current at the time they were generated.

A newer version of this spec must not retroactively invalidate prior PASS results unless:

- a previously unrecognized execution-layer failure is discovered, and
- the affected audits are formally deprecated or superseded

Such events must be explicitly documented.

13.5 Deprecation Policy

Audits may be deprecated only if:

- they are replaced by strictly stronger audits, or
- they encode guarantees no longer claimed by Symbia Seed

Deprecated audits:

- remain documented
- remain interpretable
- are clearly marked with a sunset version

13.6 Change Log (Reserved)

This section intentionally starts empty.

Future entries should include:

- date

- spec version
- summary of change
- affected audit IDs
- rationale

14. Governing Principle

If an execution-layer claim cannot be justified by emitted artifacts and explained by this document, it does not exist.

This spec exists to prevent that drift.

End of living specification.