# **Pre-Coding Essentials (Component: SECURITY.md, Version/FormulaID: VM-ENGINE v0) — 12/89**

## **1) Goal & Success**

Goal: State threat model, reporting process, and hard guarantees (offline, deterministic, no telemetry).

Success: Clear disclosure channel/SLA; users know boundaries; engineers know required security checks.

## **2) Scope**

In scope: Vulnerability disclosure, supported versions, threat model, secure-by-default config, supply-chain policy.

Out of scope: Legal licensing (in LICENSE), contribution workflow (CONTRIBUTING.md).

## **3) Inputs → Outputs**

Inputs: Engine behavior (offline, canonical JSON), CLI, schemas, fixtures, release process.

Outputs: A single SECURITY.md users can follow to report issues and operators can use to harden runs.

## **4) Entities/Tables (minimal)**

## **5) Variables (policy toggles)**

## **6) Functions**

(Doc file; no code functions.)

## **7) Algorithm Outline (sections to include)**

**Disclosure policy**

Where to report (email/PGP), info to include (version, OS, minimal repro).

Coordinated disclosure timeline; no public PoCs before fix release window.

**Supported versions**

Which tags/branches get patches; EOL policy.

**Threat model (high-level)**

**Out of scope:** network adversaries at runtime (engine is offline), multi-tenant sandboxing (single-user CLI), untrusted plugin code (none).

**In scope:** malicious or malformed local inputs; path traversal; schema bypass; report HTML injection; tie-break RNG misuse; determinism breakage; supply-chain drift.

**Hard guarantees**

No network I/O at runtime; no telemetry.

Canonical JSON (UTF-8, LF, sorted keys); exact integer/rational math; deterministic RNG (seeded) only for ties.

**Operator guidance (secure defaults)**

Run from read-only inputs directory; write outputs to separate directory.

Use --locked; prefer vendored deps; verify checksums/signatures of releases.

Provide RNG seed explicitly when tie\_policy=random; store RunRecord.

**Input handling & validation**

Enforce JSON Schema first; cross-validation (tree, magnitudes, tallies sanity).

Reject symlinks/relative ups (..) in manifest paths; resolve to canonical paths.

Max file sizes and object depth (prevent DoS); fail fast on unknown fields if strict mode enabled.

**Report rendering safety**

Reports are self-contained; no remote fonts/JS; escape all user-derived strings; sanitize HTML; content-security-policy when viewed in app.

**Build & supply chain**

Pinned toolchain; --locked; optional vendor/; signed release archives + checksums.

Third-party license review; no dynamic code download.

**Security testing**

Fuzz loaders (schemas/manifest/ballots) with structured fuzz.

Run cargo audit/cargo deny; clippy -D warnings.

Determinism test: same inputs+seed ⇒ identical RES:/RUN: IDs.

**Contact & acknowledgments**

Hall of fame/thanks section; CVE policy if applicable.

## **8) State Flow**

Reporter → email/PGP → triage (ack ≤ SLA) → fix in supported branches → coordinated disclosure → signed release with notes.

## **9) Determinism & Numeric Rules (to restate)**

No floats for comparisons; round-half-to-even only at defined points; seeded RNG recorded in RunRecord; canonical serialization for hashes.

## **10) Edge Cases & Failure Policy**

Missing seed while tie\_policy=random ⇒ reject run with clear error.

Mixed CRLF/LF or unsorted JSON in inputs ⇒ canonicalize or fail validation.

Oversized files or excessive nesting ⇒ abort with “input too large/deep”.

## **11) Test Checklist (must pass)**

Dry-run disclosure email/PGP listed and reachable.

Local run under firewall/airgap shows **zero** network connections.

Schema fuzz: no panics/UB; invalid files rejected with precise errors.

HTML report passes an XSS lint (all dynamic text escaped).

Release artifacts carry signatures/hashes; verification instructions work.