JELLEO | AUTONOMOUS SOLANA AUDIT

AUDIT CYCLE · JUNE 01, 2026

aeyakovenko/percolator (v16 engine)

AUDITOR	Kirill Sakharuk · kirill@jelleo.com
TARGET	aeyakovenko/percolator (v16 engine)
AUDIT DATE	June 01, 2026
CYCLE	`20260601-percolator-v16-liens`
ENGINE SHA	`0bee8efaea`
WRAPPER SHA	`n/a`
GENERATED	`2026-06-01T22:03:11+00:00`

0	0	2	1	0
CRITICAL	HIGH	MEDIUM	LOW	INFO

CONFIRMED · DISCLOSED · FIXED · VERIFIED

SIGNED · ED25519

MCowBQYDK2VwAyEAvCFSLBecPuNClei48PWjHuelHlBX9uYZo4wELbQ7b+k=

verify with `audit-pipeline sign verify <file> <file>.sig --pubkey jelleo.ed25519.pub`
public key at
https://jelleo.com/keys/jelleo.ed25519.pub

PLATFORM · V0.1

JELLEO · The underwriting layer for Solana DeFi.

Methodology	jelleo.com/methodology.html
Disclosure	jelleo.com/security.html
Source	github.com/Copenhagen0x/audit-pipeline-cli

Apache-2.0 · contact security@jelleo.com

00 — EXECUTIVE SUMMARY

This report documents the results of an in-depth Solana audit by Jelleo of the `aeyakovenko/percolator (v16 engine)` workspace on June 01, 2026. The cycle identified 2 Medium and 1 Low findings after Layer 2.5 triage and root-cause clustering. The findings document: (a) Resolved-mode close deadlocks on a slot-expired or impaired counterparty source-credit...; (b) Impaired insurance source-credit liens are permanently...; (c) Backing-utilization fee rounds in the account's.... Each finding is documented with affected-code citations, impact, and an LLM-authored structural fix patch. F1's resolved-close deadlock is verified by concrete execution tests (cargo test, 5/5 passing) plus code analysis — a symbolic Kani harness of the same property is provided but is CBMC-OOM at the 15 GB bound; F2 and F3 are established by code analysis. No on-chain LiteSVM reproduction was produced — the audited percolator-prog wrapper does not build/run against this engine commit — so reachability rests on concrete execution plus code-trace (see each finding's Reproduction).

00.1 — SCOPE

IN-SCOPE SOURCE SET	
Target workspace	`aeyakovenko/percolator (v16 engine)`
Protocol	Solana BPF program
Engine commit	`0bee8efaea` (0bee8efaea9ae14cedc93b939e7ee817f66c7ffb)
Source files	`src/v16.rs`
Hypothesis library	18 invariant claim(s) covering authorization, arithmetic safety, accounting consistency, capability handling, event auditability, and oracle / time freshness
Out of scope	Off-chain components (indexers, frontends, oracles); deployment scripts; framework / standard-library code; dependencies pinned in `Cargo.toml` beyond their declared interfaces.

01 — PER-FINDING ANALYSIS · CONTENTS

Each finding below begins on its own page. Numbering matches the `FINDING NN / NN` banner in the body. Click any row to jump.

FINDING 01 / 3

MEDIUM F1 resolved-close-lien-deadlock

Resolved-mode close deadlocks on a slot-expired or impaired counterparty source-credit lien

INVARIANT Resolved-mode close deadlocks on a slot-expired or impaired counterparty source-credit lien

AFFECTED CODE

- `prepare_counterparty_lien_release_delta` — `src/v16.rs:674-697`; the reject at `683-688` (`bucket.status ≠ Fresh || bucket.expiry_slot ⩽ current_slot || valid_liened_backing_num < amount`).
- `release_account_source_credit_lien_for_domain_not_atomic` — `src/v16.rs:5657-5683` (the terminal wind-down release that calls the delta above).
- `burn_account_source_claim_bound_num` — `src/v16.rs:5685-5739`; the Resolved-mode release at `5710-5714`.
- Impaired backing is write-only: `impaired_liened_backing_num` is only ever `checked_add` (`src/v16.rs:752, 761`) and is never decremented anywhere in the engine.

DESCRIPTION

Two triggers reach the same revert:

(a) Slot-expired (bucket still `Fresh`). A counterparty lien is created in Live with a future `expiry_slot` ; slots advance; the market resolves at `resolved_slot ⩾ expiry_slot` . The bucket is still `Fresh` and the backing is still in `valid_liened_backing_num` , but the `bucket.expiry_slot ⩽ current_slot` clause alone reverts the release. This is the high-reachability trigger — every counterparty lien has a finite `expiry_slot` , so any market that resolves after a lien's backing window elapses is affected.

(b) Impaired bucket. The counterparty bucket is impaired in Live via `impair_source_credit_lien_from_counterparty_not_atomic` , which migrates the backing `valid_liened → impaired_liened` and sets `status = Impaired` . There is no account-level counterparty-lien impair migration (only insurance has one), so the account still references the now-impaired lien; at resolve the `status ≠ Fresh` clause reverts the release.

IMPACT

Liveness / funds-availability (not value theft). A resolved market with any winner whose counterparty lien lapsed (expiry passed) or was impaired before resolve can never complete `close_resolved` for that account: positive-payout readiness never clears, the winner's `capital` and the vault residual are trapped, and no permissionless crank can make progress (the only Resolved-legal crank action is `Recover` , which does not release the lien).

This is reachable through the engine's ordinary lifecycle — **no special instruction is required.** The lien is created during trade settlement: a loser's realized loss becomes counterparty backing with a finite `expiry_slot` (`reserve_new_capital_backed_loss_for_source_domain_not_atomic` , on the accrual path every trade hits), and the winner's positive PnL is liened against it on the next touch (`create_account_source_credit_lien_for_effective_not_atomic`). So `Deposit → Trade → (time advances past the lien's expiry) → Resolve → Close` reaches the deadlock. Reachability and the revert are established by concrete execution tests (below) plus a full code-trace of that path. An on-chain LiteSVM witness was **not** produced: the audited `percolator-prog` wrapper does not build/run against this engine commit (0bee8ef deleted the owned-aggregate account types the wrapper was written for, and its `InitMarket` fails account-layout validation), so the evidence here is concrete-execution + code-trace, not a live transaction sequence.

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

—

LAYER 4 — ON-CHAIN BPF REPRODUCTION

—

REPRODUCTION

Established by code analysis (the gate above) plus **concrete execution** (`cargo test --features fuzz` , 5/5 passing):

- `baseline_unpatched_original_release_reverts_on_expired_lien` — on the validated slot-expired liened-winner state, the original counterparty release returns `Err(CounterUnderflow)` . This is the deadlock root, reproduced by execution (not a model abstraction).
- `fix_c1` / `fix_c2` — with the terminal-release fix, the Resolved close of the slot-expired and impaired liens both return `Ok` and drive `source_claim_liened_num` to 0 (`validate_shape` holds).
- `regression_fresh_nonexpired_close_still_succeeds` — the engine's existing `Fresh` , non-expired wind-down path (the Finding-A happy case) is unaffected.

A symbolic Kani harness of the same safety property (`proof_v16_resolved_close_deadlocks_on_expired_counterparty_lien_SHOULD_FAIL` , real `cargo kani` 0.67: symbolic `(expiry_slot, resolved_current_slot)` constrained to `expiry_slot ⩽ resolved_current_slot` , with the constructed state gated by `validate_shape` / `validate_with_market` for reachability) was authored and compiles, but the bounded model-check is **CBMC-OOM at the 15 GB memory bound** (the wide-math 256-bit division loop unwinds past the solver's capacity). The property is therefore verified by the concrete execution above rather than by a completed Kani verdict.

RECOMMENDED FIX

Add a terminal (Resolved wind-down) release that, unlike the Live-mode release, is not gated on bucket freshness/expiry: still-valid backing is returned to fresh; impaired backing is written off (not returned to fresh — it is gone) and the emptied bucket transitions to `Expired`. The Live-mode `prepare_counterparty_lien_release_delta` is left unchanged (its gate is correct outside terminal wind-down). Both branches preserve the `ReservationEncumbranceProofV16` equalities. The patch is in the fix-bundle section. Verified by execution (`cargo test`): with the fix the slot-expired and impaired Resolved closes both succeed (lien cleared, `validate_shape` holds), and the `Fresh` non-expired happy path still succeeds (no regression).

LAYER 2 — CONCRETE PROOF OF CONCEPT

No PoC source on file

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
--- a/src/v16.rs
+++ b/src/v16.rs
@@ V16Core — add a terminal (Resolved wind-down) counterparty lien release @@
 (inserted immediately before `fn prepare_counterparty_lien_release_delta`)
+    // Terminal (Resolved) wind-down counterparty lien release. Unlike the
+    // Live-mode release, this is NOT gated on bucket freshness/expiry: still-
+    // valid backing is returned to fresh; impaired backing is written off (it
+    // is gone — not returned to fresh, preserving "expiry must not increase
+    // available backing") and the emptied bucket transitions to Expired.
+    fn prepare_counterparty_lien_release_delta_terminal(
+        mut bucket: BackingBucketV16,
+        mut source: SourceCreditStateV16,
+        amount: u128,
+    ) -> V16Result<(BackingBucketV16, SourceCreditStateV16)> {
+        if amount == 0 {
+            return Ok((bucket, source));
+        }
+        if bucket.valid_liened_backing_num >= amount && source.valid_liened_backing_num >= amount {
+            bucket.valid_liened_backing_num -= amount;
+            bucket.fresh_unliened_backing_num = bucket
+                .fresh_unliened_backing_num
+                .checked_add(amount)
+                .ok_or(V16Error::CounterOverflow)?;
+            source.valid_liened_backing_num -= amount;
+        } else if bucket.impaired_liened_backing_num >= amount
+            && source.impaired_liened_backing_num >= amount
+        {
+            bucket.impaired_liened_backing_num -= amount;
+            source.impaired_liened_backing_num -= amount;
+            if bucket.fresh_unliened_backing_num == 0
+                && bucket.valid_liened_backing_num == 0
+                && bucket.impaired_liened_backing_num == 0
+            {
+                bucket.status = BackingBucketStatusV16::Expired;
+            }
+        } else {
+            return Err(V16Error::CounterUnderflow);
+        }
+        Ok((bucket, source))
+    }
+
@@ V16 view — terminal release wrapper @@
 (inserted immediately before `pub fn release_source_credit_lien_from_counterparty_not_atomic`)
+    fn release_source_credit_lien_from_counterparty_terminal_not_atomic(
+        &mut self,
+        domain: usize,
+        amount: u128,
+    ) -> V16Result<()> {
+        self.domain_asset_side(domain)?;
```

```
+        if amount == 0 {
+            return Ok(());
+        }
+        let (bucket, source) = V16Core::prepare_counterparty_lien_release_delta_terminal(
+            self.backing_bucket_for_domain(domain)?,
+            self.source_credit_for_domain(domain)?,
+            amount,
+        )?;
+        let (source, next_risk_epoch) = V16Core::prepare_source_credit_domain_recompute_for_epoch(
+            source,
+            self.header.risk_epoch.get(),
+        )?;
+        self.reservation_encumbrance_proof_for_domain_parts(
+            domain, source, bucket, self.insurance_reservation_for_domain(domain)?,
+        )?
+        .validate()?;
+        self.set_backing_bucket_for_domain(domain, bucket)?;
+        self.set_source_credit_for_domain(domain, source)?;
+        self.header.risk_epoch = V16PodU64::new(next_risk_epoch);
+        self.validate_shape()
+    }
+
@@ release_account_source_credit_lien_for_domain_not_atomic — use the terminal release in wind-down @@
         if counterparty_backing != 0 {
-            self.release_source_credit_lien_from_counterparty_not_atomic(d, counterparty_backing)?;
+            self.release_source_credit_lien_from_counterparty_terminal_not_atomic(d,
counterparty_backing)?;
```

REFERENCES

- `spec.md` line 436: "Individual liens referencing an impaired bucket become impaired by bucket status and settle later through bounded cranks" — the mandated settlement is absent.
- Engine audited at `0bee8efaea9ae14cedc93b939e7ee817f66c7ffb` (current HEAD).

FINDING 02 / 3

MEDIUM F2 impaired-lien-no-settlement

Impaired insurance source-credit liens are permanently unrecoverable — spec's recover_or_reconcile is unimplemented

INVARIANT Impaired insurance source-credit liens are permanently unrecoverable

AFFECTED CODE

- `impair_source_credit_lien_from_insurance_not_atomic` — `src/v16.rs:5522` (the only direction implemented).
- `prepare_insurance_lien_impair_delta` — `src/v16.rs:862-887` (writes `impaired_liened_insurance_num` via `checked_add` at `877, 882`).
- `release_source_credit_lien_from_insurance_not_atomic` / `consume_source_credit_lien_from_insurance_not_atomic` — `src/v16.rs:5412, 5443`; their deltas operate only on `valid_liened_insurance_num` (check `reservation.valid_liened_insurance_num < amount`, `785-792`) and cannot touch impaired state.
- No function anywhere decrements `impaired_liened_insurance_num`, `source_claim_impaired_num`, or `impaired_effective_credit_reserved`.

DESCRIPTION

Once an insurance-backed lien is impaired, the standing invariant `insurance_credit_reserved_num ≥ valid_liened_insurance_num + impaired_liened_insurance_num` (enforced at `src/v16.rs:2996` and `spec.md:583-584`) holds the reserved insurance hostage: the credit can neither be released back to `InsuranceLedger.total_available` nor consumed to settle the loss it was meant to cover. This is the insurance-side sibling of Finding 1 — both stem from impaired lien state being a write-only sink with no spec-mandated settlement path.

IMPACT

Liveness (fund-lock, not theft). Any path requiring a domain's reservations to clear — e.g. `withdraw_resolved_insurance_not_atomic`, which requires zero outstanding reservation — is blocked indefinitely once any insurance lien on that domain is impaired. Engine-level. An on-chain LiteSVM witness was not produced (same `percolator-prog`-vs-0bee8ef build limitation as Finding 1); the gap is established by code analysis (below).

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

—

LAYER 4 — ON-CHAIN BPF REPRODUCTION

—

REPRODUCTION

Established by code search: **no engine function anywhere decrements** `impaired_liened_insurance_num`, `source_claim_impaired_num`, or `impaired_effective_credit_reserved` — every write to these counters is a `checked_add` (`src/v16.rs:877, 882`), confirming the impaired state is a write-only terminal sink. The `cargo test` fix-verification confirms the round-trip once the missing recovery is added: impair → `recover_or_reconcile_impaired_insurance_lien` (`Released`) drives `impaired_liened_insurance_num` to 0 with `validate_shape` (conservation) holding.

RECOMMENDED FIX

Implement the spec transition: add `prepare_insurance_lien_recover_or_reconcile_delta` to `V16Core` and a public `recover_or_reconcile_impaired_insurance_lien_not_atomic(domain, amount, outcome)` entrypoint mirroring the existing impair/consume entrypoints (apply delta → recompute domain rate → validate `ReservationEncumbranceProofV16` and, for `Consumed`, the `TokenValueFlowProofV16` insurance-spent balance → `validate_shape`), and wire it into the Resolved wind-down / expose it as a bounded crank. The patch is in the fix-bundle section. Verified by execution (`cargo test`): impair → recover(`Released`) drives `impaired_liened_insurance_num` to 0 with `validate_shape` (conservation) holding.

LAYER 2 — CONCRETE PROOF OF CONCEPT

No PoC source on file

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
--- a/src/v16.rs
+++ b/src/v16.rs
@@ V16Core — implement spec.md:555-577 recover_or_reconcile for impaired insurance liens @@
 (inserted immediately before `fn source_credit_state_realizable_support_for_face`)
+    fn prepare_insurance_lien_recover_or_reconcile_delta(
+        mut reservation: InsuranceCreditReservationV16,
+        mut source: SourceCreditStateV16,
+        domain_spent: u128,
+        insurance: u128,
+        amount: u128,
+        consumed: bool,
+    ) -> V16Result<(InsuranceCreditReservationV16, SourceCreditStateV16, u128, u128)> {
+        if amount == 0 {
+            return Ok((reservation, source, domain_spent, insurance));
+        }
+        Self::validate_bound_num_atom_aligned(amount)?;
+        if reservation.impaired_liened_insurance_num < amount
+            || reservation.insurance_credit_reserved_num < amount
+            || source.impaired_liened_insurance_num < amount
+            || source.insurance_credit_reserved_num < amount
+        {
+            return Err(V16Error::CounterUnderflow);
+        }
+        reservation.impaired_liened_insurance_num -= amount;
+        reservation.insurance_credit_reserved_num -= amount;
+        source.impaired_liened_insurance_num -= amount;
+        source.insurance_credit_reserved_num -= amount;
+        if consumed {
+            let spend_atoms = Self::amount_from_bound_num(amount)?;
+            if insurance < spend_atoms {
+                return Err(V16Error::CounterUnderflow);
+            }
+            return Ok((
+                reservation, source,
+                domain_spent.checked_add(spend_atoms).ok_or(V16Error::CounterOverflow)?,
+                insurance - spend_atoms,
+            ));
+        }
+        Ok((reservation, source, domain_spent, insurance))
+    }
+
@@ V16 view — public entrypoint (mirrors consume_source_credit_lien_from_insurance_not_atomic) @@
 (inserted immediately before `pub fn withdraw_backing_provider_earnings_not_atomic`)
+    pub fn recover_or_reconcile_impaired_insurance_lien_not_atomic(
+        &mut self,
+        domain: usize,
+        amount: u128,
+        consumed: bool,
+    ) -> V16Result<()> {
```

```
+        self.domain_asset_side(domain)?;
+        if amount == 0 {
+            return Ok(());
+        }
+        let (reservation, source, next_domain_spent, next_insurance) =
+            V16Core::prepare_insurance_lien_recover_or_reconcile_delta(
+                self.insurance_reservation_for_domain(domain)?,
+                self.source_credit_for_domain(domain)?,
+                self.domain_insurance_budget_spent(domain)?.1,
+                self.header.insurance.get(),
+                amount,
+                consumed,
+            )?;
+        let (source, next_risk_epoch) = V16Core::prepare_source_credit_domain_recompute_for_epoch(
+            source,
+            self.header.risk_epoch.get(),
+        )?;
+        if consumed {
+            let spend_atoms = self.header.insurance.get()
+                .checked_sub(next_insurance).ok_or(V16Error::CounterUnderflow)?;
+            let vault_before = self.header.vault.get();
+            TokenValueFlowProofV16::validate_insurance_to_close_insurance_spent(
+                spend_atoms, vault_before, self.header.vault.get(),
+            )?;
+        }
+        self.reservation_encumbrance_proof_for_domain_parts(
+            domain, source, self.backing_bucket_for_domain(domain)?,
+            reservation,
+        )?
+        .validate()?;
+        self.set_insurance_reservation_for_domain(domain, reservation)?;
+        self.set_source_credit_for_domain(domain, source)?;
+        if consumed {
+            self.header.insurance = V16PodU128::new(next_insurance);
+            self.set_domain_insurance_spent(domain, next_domain_spent)?;
+        }
+        self.header.risk_epoch = V16PodU64::new(next_risk_epoch);
+        self.validate_shape()
+    }
```

REFERENCES

- `spec.md` lines 555-577 (the mandated `recover_or_reconcile_impaired_insurance_lien` transition) and line 436 (impaired liens must settle through bounded cranks).
- Engine audited at `0bee8efaea9ae14cedc93b939e7ee817f66c7ffb` (current HEAD).

FINDING 03 / 3

LOW F3 fee-rounding-direction

Backing-utilization fee rounds in the account's favor; small per-slot liens accrue zero fee

INVARIANT Backing-utilization fee rounds in the account's favor

AFFECTED CODE

- `backing_utilization_fee_quote_atoms_for_lien` — `src/v16.rs:1025-1030` (the `checked_div` truncates).
- `collect_account_backing_utilization_fee_for_domain_not_atomic` — `src/v16.rs:7701` (advances `source_lien_fee_last_slot` regardless of the floored amount).

DESCRIPTION

Two consequences follow from the floor: per-window under-collection of the truncated sub-atom fraction; and a zero-fee window — when `lien_backing_num * rate * (to_slot - from_slot) < den` (small lien, low rate, or short window) the quotient floors to `0`, and because the collector advances `source_lien_fee_last_slot` regardless, a lien collected each slot with `lien_backing_num * rate < den` accrues `0` every window indefinitely.

IMPACT

Low. The per-window leakage is dust-scale and the fee rate itself is wrapper/product policy (`spec.md:412`), so this is a rounding-direction conformance issue against `spec.md:75`, not a value drain. Reported for completeness alongside Findings 1-2.

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

—

LAYER 4 — ON-CHAIN BPF REPRODUCTION

—

REPRODUCTION

Code review against `spec.md:75` ("round against the account") and the `ceil` convention used by every other fee/notional formula in the spec (e.g. lines 99, 193-195). No PoC promoted (Low / spec-conformance).

RECOMMENDED FIX

Ceil the division and credit the truncated residue to `SettlementRoundingResidue` / `UnallocatedProtocolSurplus` through the same `TokenValueFlowProof` (`spec.md` principle 14 / line 1017), or add an explicit floor exception to `spec.md:75` for the backing-utilization fee.

LAYER 2 — CONCRETE PROOF OF CONCEPT

No PoC source on file

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

No patch authored yet

REFERENCES

- `spec.md` line 75 (round against the account), principle 14 / line 1017 (rounding-residue sink), line 412 (fee schedule is wrapper policy).
- Engine audited at `0bee8efaea9ae14cedc93b939e7ee817f66c7ffb` (current HEAD).

— A — SEVERITY RUBRIC

TIER	DEFINITION
CRITICAL	Direct loss of user funds or full protocol takeover with no meaningful preconditions. Reachable from a permissionless instruction by any signer. Must be patched immediately.
HIGH	Significant loss of user funds or protocol invariant violation under realistic preconditions (specific market state, signer with limited but obtainable role). Patch should ship in next release.
MEDIUM	Hardening issue, partial loss possible, or invariant violation requiring privileged signer or improbable state. Worth fixing in normal cadence.
LOW	Minor issue with no plausible path to fund loss. Code-quality or defense-in-depth concern.
INFO	Informational. No security impact. Documentation or style suggestion.

B — METHODOLOGY

Layer overview

LAYER	FUNCTION
`Layer 1`	Multi-agent recon. For each hypothesis, parallel LLM agents read the engine source and return a TRUE / FALSE / NEEDS_LAYER_2_TO_DECIDE verdict with confidence + per-agent grounding.
`Layer 1.5`	Adversarial debate. Contested verdicts (NEEDS_L2 or split verdicts) are promoted through a single-round attacker / defender debate, with a separate judge resolving the final verdict.
`Layer 2`	Concrete proof-of-concept. An inverted-assertion test is authored in Rust and run via `cargo test`. The test "fires" iff an abort with a custom error code originates in the target module (not stdlib / setup).
`Layer 2.5`	Triage. An LLM judge classifies each fire as `STRONG` (real bug), `SOFT` (wrong invariant), `FALSE` (artifactual abort), or `LOST` (signal missing). STRONG fires are clustered by (engine_function, target_file) so the same code-site bug under multiple hypothesis IDs collapses to one root cause.
`Layer 3`	Symbolic verification. Kani-based bounded model checking. The harness asserts the violated invariant; Kani either finds a counterexample within the bounded depth or proves safety.
`Layer 4`	On-chain BPF reproduction. The Solana program is deployed into LiteSVM and the PoC re-executed through the deployed instructions, confirming the wrapper-side defenses don't catch the bug.
`Layer P3`	Fix-bundle pipeline. The LLM authors a structural patch against the confirmed root cause and verifies it through a 6-gate machine check (well-formed diff, single-function scope, PoC fails pre-patch, PoC passes post-patch, existing tests still pass, and a language-specific symbolic/runtime check — Kani for Solana, Move Prover for Aptos, Halmos for Solidity, CBMC for C). Gates auto-skip when the language doesn’t apply (the symbolic / runtime gates of one toolchain skip on cycles authored against another, with that language’s verdict already reported under Layer 3 / Layer 4); the test-suite gate skips for eval targets that ship without a unified runner. Operator authorization is required before any upstream PR is opened.

Cycle execution

This report is a manual, in-depth audit of the percolator v16 engine; the layer pipeline in section B is Jelleo's general methodology, and the findings here were confirmed by concrete execution tests and code analysis rather than the autonomous loop. Every finding originates as a falsifiable invariant claim from a per-protocol hypothesis library, dispatched to Layer 1 multi-agent recon, promoted on contested verdicts via Layer 1.5 adversarial debate, and confirmed empirically through a Layer 2 `cargo test` proof-of-concept. Layer 2.5 triage classifies each fire as `STRONG` / `SOFT` / `FALSE` / `LOST`; only STRONG cluster representatives advance to `confirmed` and appear in §01 above. SOFT and STRONG duplicates land in `triaged`; FALSE fires return to `new`. Lifecycle: `new → triaged → confirmed → disclosed → fixed → verified`. Every cycle is signed Ed25519 against the platform key — see the cover-page receipt.

HYPOTHESES	→	POC FIRES	→	STRONG	→	ROOT CAUSES	→	CONFIRMED
18		**?**		**3**		**3**		**3**
from class library		test aborted in target module		STRONG 0 · SOFT 0 · FALSE 0 · LOST 0 (+ 3 promoted on bundle-verifier evidence)		3 STRONG → 3 after curation		reach this report

NON-FIRE ACCOUNTING 3 hypotheses were tested but the PoC did not fire — 3× `confirmed`. These are hypotheses where Layer 1 / Layer 1.5 returned a verdict but the Layer 2 PoC layer did not fire for them — they were confirmed instead by concrete execution tests and code analysis (the symbolic Kani proofs of these properties are CBMC-OOM at the 15 GB bound).

§ B.1 — Cycle funnel. Hypotheses tested → PoC fires → Layer 2.5 judge filters out artifactual / mis-invariant fires → surviving STRONG fires cluster by code site → cluster representatives become published findings.

C — AUDIT ARTIFACTS

All cycle artifacts are persisted on disk and verifiable independently of this report. The table below lists the canonical paths under the cycle workspace so a reviewer can re-execute every layer or recompute the cycle Merkle root.

ARTIFACT	PATH (RELATIVE TO WORKSPACE)
Cycle summary (manifest of every step)	`hunts/<cycle>/hunt_summary.json`
Per-step event log	`hunts/<cycle>/hunt.log.jsonl` (absent in this cycle)
Layer 2.5 triage verdicts	`hunts/<cycle>/triage.jsonl` (absent in this cycle)
Layer 2 PoC sources (Rust)	`hunts/<cycle>/poc/test_<slug>.rs` (absent in this cycle)
Layer 2 PoC run logs	`hunts/<cycle>/poc/cargo_<slug>.log` (absent in this cycle)
Layer 3 Kani harnesses + verdicts	`hunts/<cycle>/kani/<slug>/` (absent in this cycle)
Layer 4 LiteSVM exploit tests	`hunts/<cycle>/litesvm/<slug>/` (absent in this cycle)
Layer P3 fix bundles (patch.diff + evidence/ + manifest.json)	`hunts/<cycle>/bundles/<finding_id>/`
Narrative writeups (per finding)	`hunts/<cycle>/narratives/<hyp_id>.md`
Cycle Merkle root (tamper-evidence)	`hunts/<cycle>/merkle.json` (absent in this cycle)
Findings DB (SQLite)	`findings.db`
Ed25519 public key for receipt verification	`https://jelleo.com/keys/jelleo.ed25519.pub`

D — DISCLAIMERS

Findings in this report reflect the state of the engine source at the commit hash on the cover page. Subsequent changes to the codebase are not analyzed. The report is not a guarantee of code correctness or security: it documents invariants that fired (or held) under the hypothesis library applied during this cycle. Out-of-scope items are listed in §00.1 (Scope).

The findings database (see §C) reflects bundle-level state. A row is treated as a confirmed finding when the bundle's machine verification gates (PoC fails pre-patch + PoC passes post-patch + tests still pass) all hold, even if the Layer 2.5 LLM judge initially classified the fire as `SOFT` / `FALSE` / `LOST` — the verifier's

empirical patch-defuses-bug evidence supersedes the judge. Rows that did not reach a confirmed lifecycle state are retained in the findings database as audit-trail evidence but are not published findings; the authoritative set is whatever appears in §01.

Communication channel: security@jelleo.com (PGP key on jelleo.com/security.html). Coordinated disclosure follows the timeline published in our security policy; pre-disclosure leak protections are enforced at the report level (the `--public` renderer suppresses confirmed-but-not-disclosed findings).

Methodology spec: docs/methodology/ · Live reference: jelleo.com/methodology.html · Source: github.com/Copenhagen0x/audit-pipeline-cli

---