JELLEO | AUTONOMOUS SOLANA AUDIT

AUDIT CYCLE · MAY 25, 2026

percolator-bounty5-v16

AUDITOR	Kirill Sakharuk · kirill@jelleo.com
TARGET	percolator-bounty5-v16
AUDIT DATE	May 25, 2026
CYCLE	20260525-bounty5-supplementary
ENGINE SHA	23de295039
WRAPPER SHA	af8cf46
GENERATED	2026-05-25T20:47:03+00:00

CRITICAL	HIGH	MEDIUM	LOW	INFO
0	0	0	2	0

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 autonomous Solana audit cycle run by Jelleo against the `percolator-bounty5-v16` workspace on May 25, 2026. The cycle identified 2 Low findings after Layer 2.5 triage and root-cause clustering. The findings document: (a) validate_account_source_credit_shape admits empty source-domain...; (b) validate_account_shape accepts leg with rewritten asset.... Each finding includes an engine-direct proof-of-concept, a Kani-bounded model-checker proof where the formal layer ran, an on-chain BPF reproduction through LiteSVM, and an LLM-authored structural fix patch.

00.1 — SCOPE

IN-SCOPE SOURCE SET	
Target workspace	`percolator-bounty5-v16`
Protocol	Solana BPF program
Engine commit	`23de295039` (23de295039360182338e8675315103b7cf25e15b)
Source files	`(engine source enumeration unavailable in this build)`
Hypothesis library	25 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 / 2

LOW HYP1-portfolio-preflight-empty-source-domain validation_gap

Engine validate_account_source_credit_shape admits empty source-domain capacity

INVARIANT validate_account_source_credit_shape admits empty source-domain capacity

AFFECTED CODE

- Engine: src/v16.rs:11912 — validate_account_source_credit_shape
- Engine pin: 23de295039360182338e8675315103b7cf25e15b
- Wrapper guard: v16_program.rs:9437 — ensure_portfolio_storage_for_market_slots
- Wrapper pin: af8cf46

DESCRIPTION

The function reads configured_domains but never compares it against account_domain_capacity. An account whose source-domain Vec is empty (capacity 0) skips all per-domain checks and passes validation, even when the configured market requires 2 * max_market_slots domains.

Distinct from issue #104 (0x-SquidSol): that finding attacks accrue_asset_to_not_atomic K-state asymmetry to inflate pnl_pos_bound_tot and drain insurance via attacker bankruptcy. This finding is an upstream validation gap — orthogonal layer of the engine, independent root cause, no current insurance impact.

IMPACT

As-is: none observed. The wrapper auto-realloc absorbs any byte-level truncation before the engine validates.

Latent risk: if the wrapper ensure_portfolio_storage_for_market_slots is ever removed, simplified, or replaced with a check that does not trigger realloc, the engine bug becomes a direct state-corruption primitive admitting portfolios with source_domain_capacity != configured_domain_count into post-validation flow.

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

✓ Counterexample found (bug confirmed by symbolic execution)

LAYER 4 — ON-CHAIN BPF REPRODUCTION

Not reproduced (wrapper-side defenses caught it OR test setup didn't reach buggy state)

REPRODUCTION

L2 + L2.5: cargo test --features test --test test_hyp1_portfolio_preflight_lazy_deser_bypass panics with "got Ok(()) — bug present".

L3 Kani (~8.6s): cargo kani --tests --features fuzz --harness proof_hyp1_portfolio_preflight_empty_source_domain_must_reject returns VERIFICATION:- FAILED (1 of 2335 checks, 164 unreachable).

L4 LiteSVM: cargo test --test v16_cu v16_bpf_l4_hyp1_portfolio_preflight_truncated_source_domain -- --nocapture shows wrapper auto-realloc-ed truncated portfolio with zero-fill; engine bug not reached via this path.

RECOMMENDED FIX

Add an upfront check in validate_account_source_credit_shape at the top of the function: if account_domain_capacity != configured_domains return Err(V16Error::HiddenLeg). Engine becomes self-defending; the wrapper auto-realloc remains a separate defense layer.

LAYER 2 — CONCRETE PROOF OF CONCEPT (ENGINE-DIRECT)

```rust
#[test]
fn hyp1_portfolio_preflight_lazy_deser_bypass_fires() {
    // 1. Construct engine state.
    let market_group_id = [0u8; 32];
    let cfg = config_for_hyp1_portfolio_preflight_lazy_deser_bypass();
    let mut group =
        MarketGroupV16::new(market_group_id, cfg).expect("MarketGroupV16::new must succeed");

    // 2. Build a portfolio account with a well-formed header + owner, but whose
    //    source-domain vectors are empty (zero capacity) — simulating a
    //    "corrupt source-domain tail" that a lazy-deser path would not read.
    let owner: [u8; 32] = [1u8; 32];
    let account_id: [u8; 32] = [2u8; 32];
    let provenance = ProvenanceHeaderV16::new(market_group_id, account_id, owner);

    // PortfolioAccountV16::empty leaves all source-domain Vecs with len == 0,
    // which is below the configured domain count (2 assets * 2 = 4 domains).
    let account = PortfolioAccountV16::empty(provenance);

    // 3. Verify the configured domain count requires source-domain capacity.
    let required_domain_count =
        v16_domain_count_for_market_slots(cfg.max_market_slots).expect("domain count");
    // An empty PortfolioAccountV16 has source_domain_capacity() == 0.
    assert_eq!(
        account.source_claim_market_id.len(),
        0,
        "empty account has zero source-domain capacity"
    );
    assert!(
        required_domain_count > 0,
        "configured domain count must be positive"
    );

    // 4a. The "preflight" (lazy-deser path) only checks header + owner — PASSES.
    let preflight_result = preflight_owner_check(&market_group_id, &account);
    assert!(
        preflight_result.is_ok(),
        "preflight (header-only check) must accept a well-formed header: {:?}",
        preflight_result
    );

    // 4b. The full boxed read (validate_account_shape / read_portfolio_boxed_for_market_slots)
    //     checks that source_domain_capacity ≥ configured_domain_count — FAILS.
    let full_result = group.validate_account_shape(&account);
    assert!(
        full_result.is_err(),
        "full shape validation must REJECT an account whose source-domain capacity ({}) \
         is below the required domain count ({}); got Ok(()) — bug present",
        account.source_claim_market_id.len(),
        required_domain_count
    );

    // 5. The atomicity gap: preflight says Ok but full read says Err.
    //    If the engine were fixed (preflight also validated domain capacity), both
    //    would return the same error.  The assertion below captures the discrepancy.
    //    It FAILS (panics) when the bug is present because we assert the two results
    //    are EQUAL — but they are not (preflight Ok, full Err).
    let preflight_ok = preflight_owner_check(&market_group_id, &account).is_ok();
```

```
    let full_ok = group.validate_account_shape(&account).is_ok();

    // BUG WITNESS: preflight passes but full validation rejects.
    // When patched, preflight must also reject, making preflight_ok == full_ok == false.
    assert_eq!(
        preflight_ok, full_ok,
        "HYPOTHESIS CONFIRMED: lazy-deser preflight (ok={}) disagrees with full boxed \
         read (ok={}) for an account with corrupt/zero source-domain tail — \
         a wrapper that commits state based on preflight before doing the full read \
         can be bypassed",
        preflight_ok, full_ok
    );
}
```

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
--- /tmp/v16_clean.rs     2026-05-25 20:01:25.289480086 +0000
+++ /root/audit_runs/bounty5-v16/target/engine/src/v16.rs    2026-05-25 20:03:36.073811440 +0000
@@ -11912,6 +11912,13 @@
     fn validate_account_source_credit_shape(&self, account: &PortfolioAccountV16) -> V16Result<()> {
         let configured_domains = self.configured_domain_count()?;
         let account_domain_capacity = account.source_domain_capacity();
+        // SECURITY FIX (HYP1): reject portfolios whose source-domain capacity does not
+        // match the configured domain count. Previously, an account with zero-capacity
+        // source-domain Vec would pass this function unchecked because the loop body
+        // below never executes when capacity == 0.
+        if account_domain_capacity != configured_domains {
+            return Err(V16Error::HiddenLeg);
+        }
         let mut d = 0;
         while d < account_domain_capacity {
             let numeric_zero_source_domain = account.source_claim_bound_num[d] == 0
```

REFERENCES

- L3 Kani harness: tests/proofs_hyp1_portfolio_preflight_empty_source_domain.rs
- L2 PoC: tests/engine/test_hyp1_portfolio_preflight_lazy_deser_bypass.rs
- L4 LiteSVM test: target/wrapper/tests/v16_cu.rs (v16_bpf_l4_hyp1_portfolio_preflight_truncated_source_domain)

FINDING 02 / 2

LOW | 07-leg-account-ordering-not-bound | validation_gap

Engine validate_account_shape accepts leg with rewritten asset binding

INVARIANT Multi-leg oracle composite accepts oracle_accounts as a caller-supplied list. Verify each account's feed_id is bound to the expected leg index via cryptographic feed_id equality (not just account order). Find any path where leg account ordering can be swapped without detection.

AFFECTED CODE

- Engine: src/v16.rs:11791 — leg validation loop inside validate_account_shape
- Engine pin: 23de295039360182338e8675315103b7cf25e15b
- Wrapper guard: state::read_portfolio rejects malformed bytes with Custom(0x9)
- Wrapper pin: af8cf46

DESCRIPTION

The only post-attach check is that (leg.asset_index, leg.market_id) is internally consistent against self.assets[]. An attacker who can write to the on-chain PortfolioAccountV16 bytes can change both fields together without engine detection. The leg k_snap, f_snap, b_snap, epoch_snap would still reflect the original asset state but the engine would settle them against the swapped asset.

Distinct from issue #104 (0x-SquidSol): that finding attacks accrue_asset_to_not_atomic K-state asymmetry. This finding is at the validation layer — orthogonal root cause, no current insurance impact.

IMPACT

As-is: none observed. The wrapper rejects the byte-rewrite vector with Custom(0x9).

Latent risk: if exploitable, an attacker could open a position on asset A, observe the price move, then rewrite the leg to point at asset B — settling against B K/F state while having opened against A. This breaks K/F accounting invariants and could enable PnL extraction across markets.

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

✓ Counterexample found (bug confirmed by symbolic execution)

LAYER 4 — ON-CHAIN BPF REPRODUCTION

Not reproduced (wrapper-side defenses caught it OR test setup didn't reach buggy state)

REPRODUCTION

L2 + L2.5: cargo test --features test --test test_o7_leg_account_ordering_not_bound panics with "swap accepted by validate_account_shape without any cryptographic feed_id check".

L3 Kani (~74s): cargo kani --tests --features fuzz --harness proof_o7_leg_index_swap_consistent_market_id_must_reject returns VERIFICATION:- FAILED (1 of 2412 checks, 128 unreachable).

L4 LiteSVM: cargo test --test v16_cu v16_bpf_l4_o7_leg_index_rewrite_then_read -- --nocapture shows wrapper rejected byte-rewrite with Custom(0x9).

RECOMMENDED FIX

The supplied patch adds a partial mitigation: bind the leg to its asset via side-epoch. leg.epoch_snap must match the current asset epoch on the leg side (epoch_long or epoch_short).

Partial coverage: catches swaps where source and target assets have different side epochs (production scenario after retire/reactivate cycles, side resets, recovery transitions). Does NOT catch swaps between two freshly-initialized assets where both still have epoch_long = epoch_short = 0.

Full mitigation requires a dedicated binding-attestation field on PortfolioLegV16, e.g. attached_market_id_at_open: u64 captured in attach_leg and verified in validate_account_shape. That is a larger surgical change requiring synchronized updates to attach_leg, validate_account_shape, and any pathway that mutates legs.

LAYER 2 — CONCRETE PROOF OF CONCEPT (ENGINE-DIRECT)

```rust
#[test]
fn o7_leg_account_ordering_not_bound_fires() {
    let market_group_id = [0u8; 32];
    let mut group =
        MarketGroupV16::new(market_group_id, config_for_o7()).expect("MarketGroupV16::new");

    // Activate two asset slots so we have two distinct market_ids.
    {
        let asset = &mut group.assets[0];
        asset.lifecycle = AssetLifecycleV16::Active;
        asset.raw_oracle_target_price = 1_000_000;
        asset.effective_price = 1_000_000;
        asset.fund_px_last = 1_000_000;
        asset.slot_last = 1;
        asset.a_long = percolator::ADL_ONE;
        asset.a_short = percolator::ADL_ONE;
    }
    group.current_slot = group.current_slot.max(1);
    {
        let asset = &mut group.assets[1];
        asset.lifecycle = AssetLifecycleV16::Active;
        asset.raw_oracle_target_price = 2_000_000;
        asset.effective_price = 2_000_000;
        asset.fund_px_last = 2_000_000;
        asset.slot_last = 2;
        asset.a_long = percolator::ADL_ONE;
        asset.a_short = percolator::ADL_ONE;
    }
    group.current_slot = group.current_slot.max(2);

    let market_id_0 = group.assets[0].market_id;
    let market_id_1 = group.assets[1].market_id;
    assert_ne!(market_id_0, market_id_1, "assets must have distinct market_ids");

    // Build account A: long on asset 0.
    let owner_a = [1u8; 32];
    let id_a = [2u8; 32];
    let prov_a = ProvenanceHeaderV16::new(market_group_id, id_a, owner_a);
    let mut account_a = PortfolioAccountV16::empty(prov_a);
    account_a.ensure_source_domain_capacity(
        group.assets.len() * 2,
    );

    group
        .deposit_not_atomic(&mut account_a, 1_000_000_000)
        .expect("deposit A");

    let prices = vec![1_000_000u64; group.assets.len()];

    group
        .attach_leg(&mut account_a, 0, SideV16::Long, 500_000)
        .expect("attach leg to asset 0");

    // Refresh so health cert is valid.
    group
        .full_account_refresh(&mut account_a, &prices)
        .expect("refresh A");
```

```
    // Find the active leg slot for asset 0.
    let leg_slot = {
        let mut found = None;
        for s in 0..16usize {
            if account_a.legs[s].active && account_a.legs[s].asset_index == 0 {
                found = Some(s);
                break;
            }
        }
        found.expect("leg for asset 0 must exist")
    };

    // Save the original state so we can restore and compare.
    let original_asset_index = account_a.legs[leg_slot].asset_index;
    let original_market_id = account_a.legs[leg_slot].market_id;

    // Perform the swap: point the leg at asset slot 1 with asset 1's market_id.
    // This is the attacker's reordering: same leg, different oracle feed slot.
    account_a.legs[leg_slot].asset_index = 1;
    account_a.legs[leg_slot].market_id = market_id_1;

    // Invalidate health cert (the swap changed the bitmap-consistent state).
    account_a.health_cert.valid = false;

    let result_after_swap = group.validate_account_shape(&account_a);

    // Restore original state.
    account_a.legs[leg_slot].asset_index = original_asset_index;
    account_a.legs[leg_slot].market_id = original_market_id;
    account_a.health_cert.valid = false;

    let result_original = group.validate_account_shape(&account_a);

    // The original configuration must be valid.
    assert!(
        result_original.is_ok(),
        "original leg-asset binding must be valid: {:?}",
        result_original
    );

    // BUG WITNESS: if the swapped configuration is ALSO Ok, the engine has
    assert!(
        result_after_swap.is_err(),
        "BUG O7: swapping leg asset_index from {} to {} (market_id {} → {}) \
         was accepted by validate_account_shape without any cryptographic \
         feed_id check — oracle account ordering is not bound. \
         Expected Err, got Ok.",
        original_asset_index,
        1,
        original_market_id,
        market_id_1,
    );
}
```

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
--- /tmp/v16_clean.rs     2026-05-25 20:01:25.289480086 +0000
+++ /root/audit_runs/bounty5-v16/target/engine/src/v16.rs    2026-05-25 20:03:36.113811541 +0000
@@ -11800,6 +11800,18 @@
                 {
                     return Err(V16Error::HiddenLeg);
                 }
+                // SECURITY FIX (O7) — partial mitigation: bind the leg to its asset via
+                // the side-epoch snapshot. Catches attacks where the new asset has a
+                // different epoch (post-retire/reactivate, post-side-reset). Does NOT
+                // catch the case where both assets start at epoch 0 — a full mitigation
+                // requires adding a dedicated binding-attestation field on PortfolioLegV16.
+                let _asset_epoch = match leg.side {
+                    SideV16::Long => self.assets[asset_index].epoch_long,
+                    SideV16::Short => self.assets[asset_index].epoch_short,
+                };
+                if leg.epoch_snap != _asset_epoch {
+                    return Err(V16Error::HiddenLeg);
+                }
             }
         }
         if account.close_progress.active {
```

REFERENCES

- L3 Kani harness: tests/proofs_o7_leg_index_swap_consistent_market_id.rs
- L2 PoC: tests/engine/test_o7_leg_account_ordering_not_bound.rs
- L4 LiteSVM test: target/wrapper/tests/v16_cu.rs (v16_bpf_l4_o7_leg_index_rewrite_then_read)

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 cycle was produced by Jelleo's continuous, hypothesis-driven Solana audit 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.

CYCLE WALL-CLOCK `4h 0m 0s`

§ 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`
Layer 2.5 triage verdicts	`hunts/<cycle>/triage.jsonl`
Layer 2 PoC sources (Rust)	`hunts/<cycle>/poc/test_<slug>.rs`
Layer 2 PoC run logs	`hunts/<cycle>/poc/cargo_<slug>.log`
Layer 3 Kani harnesses + verdicts	`hunts/<cycle>/kani/<slug>/`
Layer 4 LiteSVM exploit tests	`hunts/<cycle>/litesvm/<slug>/`
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`
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).

§03 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 §03 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

---