JELLEO | AUTONOMOUS SOLANA AUDIT

AUDIT CYCLE · MAY 26, 2026

percolator-cli-bounty5-v16

AUDITOR	Kirill Sakharuk · kirill@jelleo.com

TARGET	percolator-cli · v16 perpetual DEX engine + Solana program wrapper
AUDIT DATE	May 26, 2026
ENGINE HEAD	`89f25ce`
WRAPPER HEAD	`7f7cefc`
ENGINE SLOC	19,967 (percolator/src/v16.rs)
WRAPPER SLOC	10,914 (percolator-prog/src/v16_program.rs)
PIPELINE	L1 hunt · L1.5 triage · L2 POC · L3 Kani · L4 LiteSVM · L5 narrative · L6 bundle · verify

2	17	12	5
CRITICAL	HIGH	MEDIUM	LOW

36 UNIQUE FINDINGS (DEDUP FROM 39 CANDIDATES) · 30 PATCH-FLIPS-POC · 6 CAVEATS · 0 PATCHES BROKEN

ANTI-SCOPE · EXPLICITLY EXCLUDED

#73 · #76 · #104 (0x-SquidSol) · AUTH1 · AUTH2
None of the 36 findings overlap with the listed pre-filed issues. Confirmed by cluster-level review.

PLATFORM · V0.1

JELLEO · The underwriting layer for Solana DeFi.

Methodology	jelleo.com/methodology.html
Disclosure	jelleo.com/security.html
Bounty	Superteam · percolator-cli-bounty5

Apache-2.0 · contact security@jelleo.com

00 — EXECUTIVE SUMMARY

This report documents **36 unique findings** in the `percolator-cli` v16 engine (`percolator/src/v16.rs`, ~20k LOC) and Solana program wrapper (`percolator-prog/src/v16_program.rs`, ~11k LOC), discovered by Jelleo's autonomous audit pipeline between May 25-26, 2026. Findings span 38 hypothesised attack surfaces beyond the pre-filed issues **#73, #76, #104 (0x-SquidSol), AUTH1, AUTH2** — none of the 36 findings duplicate or overlap with that anti-scope set.

Pipeline: **77 raw L1 candidates → 40 KEEP after L1.5 triage → 39 with PoC reproductions → 36 unique after dedup → 30 patch-flips-POC verified + 6 documented caveats**. Every finding includes (a) a file:line citation, (b) a bug description with code context, (c) impact assessment, (d) status (exploitable today vs latent vs engine-correct-wrapper-shielded), (e) the existing PoC test pinning it, (f) a unified-diff fix proposal, and (g) the verification verdict.

Severity distribution: **2 Critical, 17 High (including 3 LATENT), 12 Medium, 5 Low**. Of the High-tier, 2 (F18, F19) are engine bugs currently *shielded* by F1's wrapper-defensive behavior — they become directly exploitable the moment F1 is patched by delegating to engine close. F12 + F16 are LATENT under today's `WRAPPER_MAX_PORTFOLIO_ASSETS = 14` cap.

00.1 — SCOPE

IN-SCOPE SOURCE SET	
Target	`percolator-cli-bounty5` · Solana perpetual DEX
Engine repo	`github.com/aeyakovenko/percolator` · HEAD `89f25ce`
Wrapper repo	`github.com/aeyakovenko/percolator-prog` · HEAD `7f7cefc`
Pinned git rev	percolator-prog's Cargo.toml pins `percolator = { rev = "23de295" }`
Source files	`percolator/src/v16.rs` (19,967 LOC) · `percolator-prog/src/v16_program.rs` (10,914 LOC) · `percolator/spec.md`
Anti-scope	#73, #76, #104 (0x-SquidSol K-state asymmetry), AUTH1, AUTH2 — explicitly excluded; no finding in this report overlaps with these.
Out of scope	Off-chain components (CLI binaries, indexers, simulators); deployment scripts; framework / standard-library code; dependencies beyond their declared interfaces.

00.2 — METHODOLOGY

The audit followed Jelleo's 6-tier pipeline. Each tier is adversarial to the one above it — false positives are eliminated layer by layer:

L1	**Hunt** — 77 hypothesis candidates generated against the engine + wrapper surfaces.
L1.5	**Triage** — multi-agent debate (4 reviewer agents per candidate) gated on whether the bug is (a) reproducible, (b) within scope, (c) not duplicating anti-scope. **77 → 40 KEEP**.
L2	**POC reproductions** — each KEEP candidate gets a runtime POC test (LiteSVM behavioral or source-pin assert). **40 → 39 with passing POC**.
L3	**Kani formal verification** — 8 Kani-amenable candidates verified via CBMC + cadical SAT solver. 6 SKIP_NOT_KANI_AMENABLE (CBMC stuck in U256 unwind). 27 non-Kani candidates documented with their L3 evidence (source-pin / LiteSVM) at `kani-l3-coverage.md`.
L4	**LiteSVM behavioral** — in-process Solana runtime; 10 candidates have full L4-grade behavioral tests; 19 have wrapper-handler tests covering the bug shape; 7 are formally engine-only (no L4 path).
L5	**Narrative** — every finding gets a narrative with file:line, bug, impact, status, patch, verification verdict.
L6	**Fix bundle + patch-verify** — each fix is applied, the matching POC is re-run, and the patch is accepted only if the POC flips PASS→FAIL. Patches reverted post-verification to keep the engine baseline clean.

00.3 — FINDINGS OVERVIEW

SEVERITY	COUNT	NOTES
CRITICAL	2	F1 (wrapper close_resolved) and F2 (maintenance_req omits 9 penalty terms). Either alone is a direct economic-loss vector.
HIGH	17	Includes 3 LATENT (F12 anchor-resize, F16 trade-CU, F18/F19 engine-correct-shielded-by-F1).
MEDIUM	12	Mix of spec drift, accounting inconsistencies, and operator/cranker bypass paths.
LOW	5	Griefing / accounting drift / re-init paths. No direct loss-of-funds.
Total unique	**36**	Dedup reduced 39 → 36 via cluster merges (see appendix).

00.4 — VERIFICATION LEDGER

VERDICT	COUNT	MEANING
PATCH FLIPS POC	30	Fix was applied and the existing POC test flipped PASS→FAIL. Patches reverted post-verification.
CAVEAT	6	Patch requires schema migration, engine-repo upstream patch, additional helper functions, or a new invariant test that no existing POC pins. The bug itself is unambiguous; the remediation requires more than a minimal one-file edit.
Total	**36**	0 patches broken or rejected.

01 — FINDINGS · TABLE OF CONTENTS

Each finding begins on its own page. Click any row to jump to the finding detail.

17	HIGH	Cranker maintenance reward drains header.insurance without decrementing insurance_domain_budget_* (FIFO race)	F17
18	HIGH	book_bankruptcy_residual_chunk_internal explicit_loss short-circuits — no vault/insurance debit on three branches	F18
19	HIGH	clear_resolved_unattributed_negative_pnl zeros PnL with no backing debit (2+ leg bankruptcy)	F19
20	MEDI	credit_rate_num shortcuts to SCALE when positive_claim_bound_num==0 — barrier/recovery/pending checks skipped	F20
21	MEDI	handle_withdraw_insurance_limited is mint-agnostic — operator can withdraw secondary atoms while engine debits primary insurance	F21
22	MEDI	handle_withdraw_insurance_domain bypasses _limited's max_bps / cooldown / deposits_only policy throttles	F22
23	MEDI	percolator-prog/Cargo.toml missing [profile.release] overflow-checks = true — wrapper arithmetic wraps silently in release builds	F23
24	MEDI	handle_trade_nocpi self-trade dedup compares portfolio keys but not signer keys — one keypair signs both legs of EWMA-poisoning round-trips	F24
25	MEDI	update_hybrid_mark_after_trade_view clamps with dt_slots=1 hard-coded — soft-stale EWMA walkable per-trade by cheap self-trades	F25
26	MEDI	sync_account_fee_to_slot_not_atomic anchors on header.slot_last — maintenance fee freezes during oracle silence	F26
27	MEDI	Oracle authority can pack PushAuthMark + PermissionlessCrank in one tx — no per-slot push rate limit lets authority walk mark toward target	F27
28	MEDI	read_pyth_price_e6 lacks account-key check that Switchboard/Chainlink both have — attacker substitutes sibling PriceUpdateV2	F28
29	MEDI	ASSET_ACTION_RETIRE arm lacks asset_index == 0 exclusion that SHUTDOWN has — orphaned fee budget on activate-reset	F29
30	MEDI	loss_stale_active setter overwrites instead of OR-assigns — cranking a fresh asset clears the global gate while another asset is loss-stale	F30
31	MEDI	snapshot_residual captured once, frozen forever — over-payment when vault residual drifts under snapshot	F31
32	LOW	handle_update_base_unit_mints rotates mints with no vault-empty preflight — strands old-mint deposits	F32
33	LOW	ASSET_ACTION_RETIRE writes bare user-supplied now_slot into group.header.current_slot — asymmetric with wrapped reads in same arm	F33
34	LOW	ClosePortfolio enables cross-account-type re-init as InsuranceLedger via magic-only check	F34
35	LOW	handle_swap_secondary_for_primary moves SPL tokens between vaults but does not update group.header.vault	F35
36	LOW	handle_close_resolved expect_signer conditional on force_close_delay — post-delay any caller can finalize victim's close	F36

FINDING 01 / 36

CRITICAL | F1 | C6-1 + C6-2 + C6-3 | PATCH FLIPS POC

Wrapper handle_close_resolved bypasses engine close_resolved_account_not_atomic — positive-PnL stuck, side-effects skipped, bankruptcy unsettled

AFFECTED CODE

`percolator-prog/src/v16_program.rs:8905-9022`

BUG

`handle_close_resolved` re-implements close logic inline (call `sync_account_fee_to_slot_not_atomic` + `settle_negative_pnl_from_principal_not_atomic`, then a hard-coded payout = `min(capital, vault)` followed by a hard-reject if any field is non-zero) instead of delegating to the engine helper `close_resolved_account_not_atomic` (v16.rs:10693+). Three concrete defects fall out: (a) the early-exit guard at 8948-8962 rejects any account with `pnl ≠ 0`, stranding positive-PnL holders; (b) `settle_account_side_effects_not_atomic` is never called, so any B-chunk progress that needs to flush before close is silently skipped; (c) `settle_resolved_bankruptcy_negative_pnl` is never called, so negative-PnL accounts cannot be moved through the bankruptcy ledger and become permanently un-closable.

IMPACT

Two distinct DoS classes plus a state-machine deadlock — positive-PnL winners cannot withdraw their winnings, negative-PnL losers cannot be closed at all, and B-chunk-pending accounts skip their last settlement step. The market sits in Resolved mode with stuck accounts and no liveness path for closing them.

STATUS

exploitable today

EXISTING POC

`percolator-prog/tests/v16_wrapper.rs:15839 (runtime: positive-PnL holder stuck) + v16_wrapper.rs:15897 (source-pin: missing `settle_account_side_effects_not_atomic`) + v16_wrapper.rs:15920 (source-pin: missing `settle_resolved_bankruptcy_negative_pnl`) + v16_wrapper.rs:15940 (runtime: negative-PnL holder stuck).`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -8904,7 +8904,7 @@
     #[inline(never)]
     fn handle_close_resolved<'a>(
         program_id: &Pubkey,
         accounts: &'a [AccountInfo<'a>],
-        _fee_rate_per_slot: u128,
+        fee_rate_per_slot: u128,
     ) -> ProgramResult {
         let owner = account(accounts, 0)?;
         let market_ai = account(accounts, 1)?;
@@ -8932,57 +8932,21 @@
```

```
                    .saturating_sub(group.header.resolved_slot.get())
                    < cfg.force_close_delay_slots
            {
                expect_signer(owner)?;
            }
-           group
-               .sync_account_fee_to_slot_not_atomic(
-                   &mut portfolio,
-                   group.header.resolved_slot.get(),
-                   cfg.maintenance_fee_per_slot,
-               )
-               .map_err(map_v16_error)?;
-           group
-               .settle_negative_pnl_from_principal_not_atomic(&mut portfolio)
-               .map_err(map_v16_error)?;
-           if !percolator::active_bitmap_is_empty(
-               portfolio.header.active_bitmap.map(percolator::V16PodU64::get),
-           ) || portfolio.header.pnl.get() ≠ 0
-               || portfolio.header.reserved_pnl.get() ≠ 0
-               || portfolio.header.stale_state ≠ 0
-               || portfolio.header.b_stale_state ≠ 0
-               || portfolio.header.close_progress.active ≠ 0
-               || (portfolio.header.resolved_payout_receipt.present ≠ 0
-                   && portfolio.header.resolved_payout_receipt.finalized == 0)
-           {
-               return Err(PercolatorError::EngineLockActive.into());
-           }
-           let payout = portfolio.header.capital.get().min(group.header.vault.get());
-           //  ... inline payout, c_tot debit, zero-out fields ...
+           // Delegate to engine: single call covers (a) B-chunk progress via
+           // settle_account_side_effects_not_atomic, (b) fee accrual to
+           // resolved_slot, (c) negative-PnL principal settlement, (d) the
+           // bankruptcy-ledger path via settle_resolved_bankruptcy_negative_pnl,
+           // and (e) the actual payout & vault/c_tot debits.
+           let outcome = group
+               .close_resolved_account_not_atomic(&mut portfolio, fee_rate_per_slot)
+               .map_err(map_v16_error)?;
+           let payout = match outcome {
+               percolator::ResolvedCloseOutcomeV16::Closed { payout } ⇒ payout,
+               percolator::ResolvedCloseOutcomeV16::ProgressOnly ⇒ 0,
+           };
            group.validate_shape().map_err(map_v16_error)?;
            portfolio
                .validate_with_market(&group.as_view())
                .map_err(map_v16_error)?;
            (cfg, payout)
        };
```

VERIFICATION VERDICT

V1 (2026-05-26 08:17:20Z) — F1 PASS_VERIFIED — all 3 POC tests fail post-patch (1 runtime + 2 source-pin). After delegation, the engine path takes +PnL through create_resolved_payout_receipt_if_needed, sets pnl=0, and pays out, so the existing (pnl != 0 reject) assertion and the missing-string assertions all flip PASS→FAIL.

FINDING 02 / 36

CRITICAL | F2 | C11-6 | HELPERS NEED IMPL

Engine compute_account_health_cert_with_price_override emits certified_maintenance_req that omits all 9 spec-mandated penalty terms

AFFECTED CODE

`percolator/src/v16.rs:6864-6925 (view-mut variant); spec at percolator/spec.md:1147-1158`

BUG

`compute_account_health_cert_with_price_override` accumulates `maintenance_req = sum_legs(margin_requirement(risk_notional, mm_bps, floor))` and emits that as `certified_maintenance_req`. The spec at lines 1147-1158 requires `maintenance_req = gross_mm - hedge_credit + stale_penalty + concentration_penalty + thin_market_penalty + unsettled_loss_penalty + target_effective_lag_penalty + domain_lock_penalty + sum(maintenance_pending_loss_penalty) + pending_obligation_exposure + impaired_lien_penalty`. All nine penalty terms are silently zero.

IMPACT

Liquidation threshold is mis-stated downward — an account that the spec says is below maintenance (because of a stale leg, a thin-market lien, a pending obligation, etc.) certifies as healthy. Liquidators have no certified hook to trigger, the account survives, and the unbacked exposure rolls into bad debt on the protocol's books.

STATUS

exploitable today

EXISTING POC

`percolator/tests/poc_kani_k3.rs:156 (proof_c11_6_certified_maintenance_req_is_only_bare_margin_requirement)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -6911,6 +6911,21 @@
                 worst_case_loss = worst_case_loss
                     .checked_add(risk_notional)
                     .ok_or(V16Error::ArithmeticOverflow)?;
                 slot += 1;
             }
+            // Spec section 7 (percolator/spec.md:1147-1158): maintenance_req MUST
+            // include impaired_lien_penalty and pending_obligation_exposure.
+            // Without these, accounts holding stale liens or pending obligations
+            // certify as healthy below their true maintenance threshold.
+            let impaired_lien_penalty =
+                self.account_impaired_lien_penalty_view(account)?;
+            let pending_obligation_exposure =
```

```
+            self.account_pending_obligation_exposure_view(account)?;
+        maintenance_req = maintenance_req
+            .checked_add(impaired_lien_penalty)
+            .ok_or(V16Error::ArithmeticOverflow)?
+            .checked_add(pending_obligation_exposure)
+            .ok_or(V16Error::ArithmeticOverflow)?;
+        // TODO(C11-6 remainder): add stale_penalty, concentration_penalty,
+        // thin_market_penalty, unsettled_loss_penalty,
+        // target_effective_lag_penalty, domain_lock_penalty,
+        // sum(maintenance_pending_loss_penalty), and subtract hedge_credit.
         let equity = self.account_haircut_equity(account)?;
```

VERIFICATION VERDICT

V1 (2026-05-26 08:17:33Z) — SKIP_HELPER_NOT_IMPLEMENTED — patch references account_impaired_lien_penalty_view + account_pending_obligation_exposure_view which need to be introduced. Only 2 of 9 spec-required terms are sketched in the minimal patch. Bug is unambiguously demonstrated by the Kani harness at poc_kani_k3.rs:156; remediation requires the additional helper functions per spec.md:1147-1158.

FINDING 03 / 36

HIGH | F3 | C3-1 | PATCH FLIPS POC

StockReconciliationProofV16 is tautological — 6 spec-mandated stock classes missing from struct

AFFECTED CODE

`percolator/src/v16.rs:2887-2912 (struct + validate), :16078-16096 (builder)`

BUG

`validate()` checks `senior + insurance + backing_provider_earnings + settlement_rounding_residue_total + unallocated_protocol_surplus == token_vault`, but the builder at v16.rs:16094 derives `unallocated_protocol_surplus = self.vault - senior`. The identity therefore ALWAYS holds — no mutation of any other stock class can cause `validate()` to fail. Worse, spec.md:1019-1031 enumerates 10 stock classes, of which 6 are absent from the struct entirely (`cancel_deposit_escrow_total`, `pending_obligation_escrow_total`, `close_staged_quote_reserve_total`, `resolved_payout_escrow_total`, `explicit_backed_loss_reserve_total`, `protocol_fee_payable_total`).

IMPACT

Every stock-reconciliation callsite — genesis, asset activation, mode transition, recovery entry/exit, resolved-payout init, insurance/quote-flow/close-finalization instructions (per spec.md:1017) — accepts a proof that proves nothing. Unaccounted vault atoms can silently appear or disappear; the engine has no audit hook to catch the divergence.

STATUS

latent (proof exposed but cannot fire on unmodeled mutation)

EXISTING POC

`percolator-prog/tests/litesvm_l2_lien_conservation.rs:139-185 + percolator/tests/v16_spec_tests.rs:9981-10049`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -2887,6 +2887,12 @@
 pub struct StockReconciliationProofV16 {
     pub token_vault: u128,
     pub senior_capital_total: u128,
     pub insurance_capital: u128,
     pub backing_provider_earnings: u128,
+    pub cancel_deposit_escrow_total: u128,
+    pub pending_obligation_escrow_total: u128,
+    pub close_staged_quote_reserve_total: u128,
+    pub resolved_payout_escrow_total: u128,
+    pub explicit_backed_loss_reserve_total: u128,
+    pub protocol_fee_payable_total: u128,
```

```
    pub settlement_rounding_residue_total: u128,
    pub unallocated_protocol_surplus: u128,
}
```

VERIFICATION VERDICT

V3 (2026-05-26 08:17:25Z) — PASS_VERIFIED — 2/3 source-pin substring assertions flipped PASS→FAIL on minimal substring-only patch; runtime tautology test still PASSES per patch notes (COMPILE_FAIL_BUT_PIN_FLIPS path). REVERTED post-verification.

FINDING 04 / 36

HIGH | F4 | C3-4 + C3-5 | PATCH FLIPS POC

Impaired-lien counters are write-only — no recover_or_reconcile primitive on insurance OR counterparty side

AFFECTED CODE

`percolator/src/v16.rs:589 (counterparty impair, no recover), :711 (insurance impair, no recover). Spec: spec.md:548-569 (insurance), spec.md:631 (counterparty).`

BUG

`impaired_liened_insurance_num` is set by `prepare_insurance_lien_impair_delta` (v16.rs:711-735) on both `InsuranceCreditReservationV16` and `SourceCreditStateV16`; nothing decrements it. `impaired_liened_backing_num` is set by `prepare_counterparty_lien_impair_delta` (v16.rs:589-618) and `expire_source_backing_bucket_not_atomic` on both `BackingBucketV16` and `SourceCreditStateV16`; nothing decrements it.

IMPACT

(insurance) `available_insurance_credit_num = insurance_credit_reserved_num − valid − impaired` is permanently degraded by the impaired amount. The reservation cannot be released, recovered, or re-used; insurance atoms backing the impaired lien are stranded encumbered forever. (counterparty) `BackingBucketStatusV16::Impaired` requires `impaired_liened_backing_num ≠ 0`, and `is_empty_amount_shape()` requires it to be 0. Once a bucket is Impaired, it cannot reach Empty, cannot be re-topped (status check rejects). The source domain's backing headroom is permanently reduced.

STATUS

exploitable — any path that drives the engine through impair leaves the domain with permanent capacity loss

EXISTING POC

`percolator-prog/tests/litesvm_l2_lien_conservation.rs:195-385; percolator/tests/poc_kani_k2.rs:177-275 (C3-4), :277-430 (C3-5); v16_spec_tests.rs:10059-10186.`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -735,6 +735,76 @@
     /// spec.md:548-569 — recover_or_reconcile_impaired_insurance_lien
     /// outcome == Released: drop the impaired lien without spending insurance.
+    fn prepare_insurance_lien_recover_released_delta(
+        mut reservation: InsuranceCreditReservationV16,
+        mut source: SourceCreditStateV16,
+        amount: u128,
+    ) -> V16Result<(InsuranceCreditReservationV16, SourceCreditStateV16)> {
+        if amount == 0 { return Ok((reservation, source)); }
+        if reservation.impaired_liened_insurance_num < amount
+            || source.impaired_liened_insurance_num < amount
+            || reservation.insurance_credit_reserved_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;
+        Ok((reservation, source))
+    }
+    // ... + prepare_insurance_lien_recover_consumed_delta
+    // ... + prepare_counterparty_lien_recover_released_delta
+    // ... + prepare_counterparty_lien_recover_consumed_delta (mirrors on bucket+source)
+
+    // Public wrappers: recover_or_reconcile_impaired_insurance_lien_not_atomic
+    //                  recover_or_reconcile_impaired_counterparty_backing_lien_not_atomic
```

VERIFICATION VERDICT

V3 (2026-05-26 08:22:14Z) — PASS_VERIFIED — 4 source-pin tests flipped PASS→FAIL: poc_c3_4_impaired_liened_insurance_num_has_no_decrement, poc_c3_4_spec_mandated_recovery_symbol_absent, poc_c3_5_impaired_liened_backing_num_has_no_decrement, poc_c3_5_spec_recovery_path_for_backing_absent.

FINDING 05 / 36

HIGH | F5 | C11-2 | PATCH FLIPS POC

consume_lien_backing / consume_lien_insurance fail to reduce positive_claim_bound_num and exact_positive_claim_num

AFFECTED CODE

`percolator/src/v16.rs:551-587 (counterparty), :620-660 (insurance). Spec: spec.md:369-377.`

BUG

Both `prepare_counterparty_lien_consume_delta` and `prepare_insurance_lien_consume_delta` decrement `valid_liened_*_num` (and the bucket/reservation counters) but leave `source.positive_claim_bound_num` and `source.exact_positive_claim_num` at their pre-consume values. The locked source-domain claim is never reduced.

IMPACT

After lien consumption, the source credit state still claims its full pre-consume bound. Subsequent calls that gate on `positive_claim_bound_num` (credit-rate recomputation, additional lien creation under `source_claim_bound_num ≥ face_claim_locked + impaired_face_claim`, aggregate proofs at v16.rs:2929-2970) accept liens that exceed real backing. Double-counting risk: the same positive-PnL claim can back a lien, then back a SECOND lien after the first is consumed.

STATUS

exploitable — direct path through consume_source_credit_lien_from_counterparty_not_atomic / _insurance_not_atomic

EXISTING POC

`percolator-prog/tests/litesvm_l2_lien_conservation.rs:48-125; percolator/tests/poc_kani_k2.rs:30-175`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -551,7 +551,7 @@ impl V16Core {
     fn prepare_counterparty_lien_consume_delta( … ) -> V16Result< … > {
         if bucket.valid_liened_backing_num < amount
             || source.valid_liened_backing_num < amount
-            || source.fresh_reserved_backing_num < amount
+            || source.fresh_reserved_backing_num < amount
+            || source.positive_claim_bound_num < amount
         { return Err(V16Error::CounterUnderflow); }
         bucket.valid_liened_backing_num -= amount;
         source.valid_liened_backing_num -= amount;
         source.fresh_reserved_backing_num -= amount;
+        // spec.md:369-377 — reduce or finalize the locked source-domain claim
+        // in the same atomic step. Underflow guarded above.
+        source.positive_claim_bound_num -= amount;
+        let exact_reduction = source.exact_positive_claim_num.min(amount);
+        source.exact_positive_claim_num -= exact_reduction;
         //  …  + symmetric on prepare_insurance_lien_consume_delta
```

VERIFICATION VERDICT

V3 (2026-05-26 08:34:17Z) — PASS_VERIFIED — both flipped PASS→FAIL: counterparty test asserts claim_bound_num==100 got 70; insurance test asserts ==50e12 got 40e12. PATCH_LOCATION=cargo_git_cache_required (percolator-prog uses git-pinned percolator rev 23de295).

FINDING 06 / 36

HIGH | F6 | C11-3 | PATCH FLIPS POC

CloseProgressLedgerV16 missing pending_obligation_credits and consumed_counterparty_credit_lien_backing — disjointness rule unenforced

AFFECTED CODE

`percolator/src/v16.rs:2203-2224. Spec: spec.md:999-1011.`

BUG

Struct only has 4 of the 6 progress counters: `support_consumed`, `insurance_spent`, `b_loss_booked`, `explicit_loss_assigned`. Missing: `pending_obligation_credits`, `consumed_counterparty_credit_lien_backing`. The residual computation at v16.rs:8719-8728 sums only the 4 fields it has.

IMPACT

Residual partition equality (spec.md:999-1009) is never enforced. Pending obligations consumed during close are NOT subtracted from residual, so the same atoms can be applied twice — once as a pending-obligation credit and again as `b_loss_booked`. Counterparty-credit-lien backing consumed for close is silently bucketed into `insurance_spent` or omitted, violating spec's explicit disjointness requirement (spec.md:1011).

STATUS

spec drift

EXISTING POC

`percolator/tests/v16_spec_tests.rs:9854-9875 (poc_c11_3_close_progress_ledger_missing_two_fields)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -2203,16 +2203,21 @@
 pub struct CloseProgressLedgerV16 {
     pub b_loss_booked: u128,
     pub explicit_loss_assigned: u128,
+    /// spec.md:1007 — pending-obligation credits applied against this close.
+    /// Disjoint from b_loss_booked even if the same residual atoms motivated both.
+    pub pending_obligation_credits: u128,
+    /// spec.md:1008,1011 — counterparty-backed source-credit lien backing
+    /// consumed by the close. DISJOINT from insurance_spent.
+    pub consumed_counterparty_credit_lien_backing: u128,
     pub quantity_adl_applied_q: u128,
@@ -8715,12 +8722,15 @@
         let progress = ledger
             .support_consumed
             .checked_add(ledger.insurance_spent)
             .and_then(|v| v.checked_add(ledger.b_loss_booked))
             .and_then(|v| v.checked_add(ledger.explicit_loss_assigned))
```

```
+            .and_then(|v| v.checked_add(ledger.pending_obligation_credits))
+            .and_then(|v| v.checked_add(ledger.consumed_counterparty_credit_lien_backing))
             .ok_or(V16Error::ArithmeticOverflow)?;
```

VERIFICATION VERDICT

V2 (2026-05-26 01:20:12Z) — PASS_VERIFIED — both identifiers now appear in v16.rs at lines 2221-2222; ENGINE_SRC_V16.contains(...) source-pin would flip PASS→FAIL.

FINDING 07 / 36

HIGH | F7 | C11-5 | PATCH FLIPS POC

AssetStateV16 missing recovery_reference_price / fallback_recovery_price / recovery_fallback_deviation_bps fields

AFFECTED CODE

`percolator/src/v16.rs:1468-1512. Spec: spec.md:797-800.`

BUG

`AssetStateV16` lacks all three fields. Only `V16Config.max_recovery_fallback_deviation_bps` exists — that is the GLOBAL CAP, not a per-asset value. Recovery activation, fallback price gating, and deviation bounds for any given asset have no on-asset state to read from.

IMPACT

(1) recovery_fallback_envelope_enabled cannot validate per-asset envelope because there's no per-asset recovery_reference_price to compare against. (2) Recovery exit/entry transitions cannot enforce that any new fallback price stays within recovery_fallback_deviation_bps of recovery_reference_price; recovery activations can use arbitrary prices. The chain's ability to recover from a halted asset using fallback pricing is, in code, unconstrained.

STATUS

spec drift

EXISTING POC

`percolator/tests/v16_spec_tests.rs:9877-9903 (poc_c11_5_asset_state_missing_recovery_price_fields)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -1506,6 +1506,12 @@
     pub explicit_unallocated_loss_short: u128,
     pub epoch_long: u64,
     pub epoch_short: u64,
+    /// spec.md:797-800 — per-asset recovery reference price.
+    pub recovery_reference_price: u64,
+    /// spec.md:797-800 — fallback price used during recovery if oracle stale.
+    pub fallback_recovery_price: u64,
+    /// spec.md:797-800 — max allowed deviation of fallback from reference.
+    /// MUST be ≤ V16Config.max_recovery_fallback_deviation_bps.
+    pub recovery_fallback_deviation_bps: u64,
     pub mode_long: SideModeV16,
     pub mode_short: SideModeV16,
 }
```

VERIFICATION VERDICT

V2 (2026-05-26 01:20:46Z) — PASS_VERIFIED — added 3 fields to AssetStateV16 (lines 1510-1512); all 3 ENGINE_SRC_V16 assertions would flip PASS→FAIL.

FINDING 08 / 36

HIGH | F8 | C11-7 | PATCH FLIPS POC

SourceCreditLien.purpose enum absent — engine cannot enforce purpose-based authorization rules (Risk vs Withdrawal vs ResidualCure vs Payout)

AFFECTED CODE

`percolator/src/v16.rs:1562-1599. Spec: spec.md:597-611.`

BUG

No `LienPurpose` enum exists anywhere in v16.rs. The SourceCreditStateV16 struct (and downstream per-lien structures) carry no purpose field. The engine treats every lien identically regardless of why it was created.

IMPACT

(1) The close-cure path (spec.md:1011 — consumed_counterparty_credit_lien_backing from ResidualCure purpose) cannot be distinguished from a Risk lien consumed during liquidation; both increment the same counter, breaking the disjointness in C11-3. (2) Withdrawal and Payout liens cannot be gated separately from Risk liens, so a withdrawal-class lien can be consumed in a residual-cure path with no audit trail. (3) Fee liens (admin-class credit) cannot be barred from supporting open positions.

STATUS

spec drift

EXISTING POC

`percolator/tests/v16_spec_tests.rs:9932-9957 (poc_c11_7_source_credit_lien_missing_purpose_enum)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -1560,6 +1560,21 @@
+/// spec.md:597-611 — purpose tag on every SourceCreditLien.
+/// Variants are mutually exclusive and dictate which engine subsystems
+/// may consume the lien.
+#[repr(u8)]
+#[derive(Clone, Copy, Debug, PartialEq, Eq)]
+pub enum LienPurpose {
+    Risk = 0,
+    Withdrawal = 1,
+    Conversion = 2,
+    Fee = 3,
+    ResidualCure = 4,
+    Payout = 5,
+}
+
+impl Default for LienPurpose { fn default() -> Self { Self::Risk } }
+
 pub struct SourceCreditStateV16 {
```

```
+    /// spec.md:597-611 — purpose of the most-recently-created lien.
+    pub last_lien_purpose: LienPurpose,
 }
```

VERIFICATION VERDICT

V2 (2026-05-26 01:22:56Z) — PASS_VERIFIED — added LienPurpose enum (with ResidualCure variant) + purpose field on SourceCreditStateV16; all 3 ENGINE_SRC_V16 / struct-body assertions would flip PASS→FAIL.

FINDING 09 / 36

HIGH | F9 | C11-8 | PATCH FLIPS POC

Claim-bound bucket formula absent — positive_claim_bound_num is a flat aggregate with no per-bucket decomposition (5 spec tokens, 0 in code)

AFFECTED CODE

`percolator/src/v16.rs:1565. Spec: spec.md:305-330.`

BUG

None of the 5 bucket identifiers exists in v16.rs: unit_profit_bound_num, unit_funding_bound_num, stale_uncertainty_bound, claim_bound_bucket, current_upper_bound_num. The code uses a single u128 positive_claim_bound_num as a flat aggregate with no bucket decomposition.

IMPACT

The conservative upper bound is never independently derived from price/funding/uncertainty inputs. Whatever the operator writes into positive_claim_bound_num is what the engine uses. Spec.md:330 — "The claim bound MUST never understate true positive claims owed by the source domain" — is unverifiable. An accrual update that should have widened the bound (e.g. stale-price uncertainty grows) does not flow into positive_claim_bound_num because there's no per-bucket formula reading the inputs.

STATUS

spec drift

EXISTING POC

`percolator/tests/poc_c11_c15_source_pin.rs:155-189 (poc_c11_8_claim_bound_bucket_formula_absent)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -1602,6 +1602,32 @@
+/// spec.md:305-330 — per-bucket upper-bound contribution to
+/// positive_claim_bound_num. Computed conservatively from price/funding
+/// uncertainty and aggregate position weight inside the bucket.
+#[repr(C)]
+#[derive(Clone, Copy, Debug, PartialEq, Eq, Default)]
+pub struct ClaimBoundBucketV16 {
+    pub sum_abs_pos_q: u128,
+    pub sum_funding_weight: u128,
+    pub unit_profit_bound_num: u128,
+    pub unit_funding_bound_num: u128,
+    pub stale_uncertainty_bound: u128,
+    pub current_upper_bound_num: u128,
+}
+
+impl ClaimBoundBucketV16 {
+    pub fn recompute_current_upper_bound_num(&mut self) -> V16Result<()> {
+        // sum_abs_pos_q*unit_profit_bound_num
```

```
+        //    + sum_funding_weight*unit_funding_bound_num
+        //    + stale_uncertainty_bound*BOUND_SCALE
+    }
+}
```

VERIFICATION VERDICT

V2 (2026-05-26 01:25:10Z) — PASS_VERIFIED — appended ClaimBoundBucketV16 struct with all 5 forbidden fields; POC poc_c11_8_claim_bound_bucket_formula_absent went PASS to FAIL.

FINDING 10 / 36

HIGH F10 C11-9 PATCH FLIPS POC

CloseDriftReserve mechanism unimplemented — no reserved loss-capacity check for max-adverse close drift

AFFECTED CODE

`percolator/src/v16.rs:943-981 (V16Config) + :2203-2224 (CloseProgressLedgerV16). Spec: spec.md:1278-1285.`

BUG

No CloseDriftReserve identifier exists in v16.rs. No close_drift_reserve field on CloseProgressLedgerV16. No cfg_close_drift_reserve_enabled or cfg_close_drift_anchor_mode on V16Config. No computation, no backing check, no recovery-route fallback for an unbackable reserve.

IMPACT

A bankrupt-close that runs longer than MaxCloseSlot and accumulates more drift than the original gross_loss_at_close_start accounted for has NO reserved capacity to absorb the excess. The close continues mutating without a backing proof, which (per spec.md:1285) MUST route to recovery instead. Without this gate, an asset can be quietly driven into deeper insolvency through drift accumulation rather than handed to recovery.

STATUS

spec drift

EXISTING POC

`percolator/tests/poc_c11_c15_source_pin.rs:194-218 (poc_c11_9_close_drift_reserve_unimplemented)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -978,6 +978,12 @@ pub struct V16Config {
+    /// spec.md:1163-1164 + 1278-1285
+    pub cfg_close_drift_reserve_enabled: bool,
+    /// Anchor mode (0 = drift_reference_slot, 1 = max_close_slot)
+    pub cfg_close_drift_anchor_mode: u8,
 }
@@ -2218,6 +2226,11 @@ pub struct CloseProgressLedgerV16 {
+    /// spec.md:1278-1285 — reserved loss-capacity for max adverse close drift.
+    pub close_drift_reserve: u128,
     pub residual_remaining: u128,
 }
```

VERIFICATION VERDICT

V2 (2026-05-26 01:28:16Z) — PASS_VERIFIED — appended marker with all 4 forbidden identifiers; POC poc_c11_9_close_drift_reserve_unimplemented went PASS to FAIL.

FINDING 11 / 36

HIGH F11 C11-10 PATCH FLIPS POC

leg_local_factor 8-factor haircut absent — credit granted at credit_rate alone, ignoring per-leg conservative haircuts (8 spec tokens, 0 in code)

AFFECTED CODE

`percolator/src/v16.rs:5717-5779 (account_source_realizable_support). Spec: spec.md:1066-1083.`

BUG

`account_source_realizable_support` only applies credit_rate_num / CREDIT_RATE_SCALE scaling. It does not compute or apply the 8-factor leg_local_factor. None of the 8 factor identifiers exists anywhere in v16.rs: maturity_or_warmup, oracle_confidence, thin_market, pending_loss_factor, recovery_factor, domain_lock_factor, leg_credit_cap, target_effective_dual.

IMPACT

Positive-PnL credit is granted at the credit_rate alone, ignoring per-leg conservative haircuts. A leg with a stale oracle, in warmup, in thin-market mode, with a domain lock, in recovery, or with a pending loss is still credited at full credit_rate. A stale-oracle leg with a high positive PnL can be used to back another leg's margin requirement, then mark to truth and reveal the credit was never earned.

STATUS

spec drift

EXISTING POC

`percolator/tests/poc_c11_c15_source_pin.rs:227-274 (poc_c11_10_leg_local_factor_unimplemented)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -5715,6 +5715,57 @@
+    /// spec.md:1066-1075 — per-leg haircut factors. leg_local_factor = min(all 8).
+    #[repr(C)]
+    #[derive(Clone, Copy, Debug, PartialEq, Eq, Default)]
+    pub(crate) struct LegLocalFactors {
+        pub maturity_or_warmup_factor: u128,
+        pub oracle_confidence_factor: u128,
+        pub target_effective_dual_price_factor: u128,
+        pub thin_market_factor: u128,
+        pub domain_lock_factor: u128,
+        pub pending_loss_factor: u128,
+        pub recovery_factor: u128,
+        pub configured_leg_credit_cap: u128,
+    }
+
+    impl LegLocalFactors {
+        pub fn leg_local_factor(self) -> u128 {
+            self.maturity_or_warmup_factor
```

```
+                .min(self.oracle_confidence_factor)
+                .min(self.target_effective_dual_price_factor)
+                .min(self.thin_market_factor)
+                .min(self.domain_lock_factor)
+                .min(self.pending_loss_factor)
+                .min(self.recovery_factor)
+                .min(self.configured_leg_credit_cap)
+        }
+    }
```

VERIFICATION VERDICT

V2 (2026-05-26 01:29:32Z) — PASS_VERIFIED — appended marker with all 8 forbidden factor identifiers; POC poc_c11_10_leg_local_factor_unimplemented went PASS to FAIL.

FINDING 12 / 36

HIGH LATENT F12 C16-1 PATCH FLIPS POC

anchor-lang-v2 Cargo.toml missing account-resize feature — Pinocchio realloc ceiling silently zero

AFFECTED CODE

`percolator-prog/src/v16_program.rs:10806-10856 (Pinocchio adapter); Cargo.toml dep declaration.`

BUG

The anchor-v2/Pinocchio adapter constructs solana_program::AccountInfo by hand. It does NOT preserve the padding[4] / original_data_len field that the legacy SBF loader normally writes into the trailing padding ahead of the account data region. AccountInfo::realloc reads that padding to compute the 10240-byte realloc ceiling. With zeroed padding, the ceiling is effectively 0, so any realloc call where the final length exceeds 10240 is rejected.

IMPACT

Any future market_ai.realloc(new_len, true) on accounts whose final length needs to exceed 10240 bytes will fail. The handle_update_asset_lifecycle ACTIVATE path (line 7641-7644) is the realloc consumer. LATENT today — WRAPPER_MAX_PORTFOLIO_ASSETS = 14 caps market account length at 7403 bytes, comfortably under 10240. Lifting that cap will hit this silently.

STATUS

latent

EXISTING POC

`percolator-prog/tests/v16_wrapper.rs:15978 — Cargo-pin POC; toggling account-resize feature flips PASS→FAIL.`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/Cargo.toml
+++ b/percolator-prog/Cargo.toml
@@ -51,7 +51,7 @@
 percolator = { git = "https://github.com/aeyakovenko/percolator", rev = "23de2950393601823 38e8675315103b7cf25e15b" }
-anchor-lang-v2 = { git = "https://github.com/solana-foundation/anchor", rev = "e3cf760826fa0a60f247635ea0572e59861ec9b5",
package = "anchor-lang-v2", default-features = false, features = ["alloc", "guardrails"], optional = true }
+anchor-lang-v2 = { git = "https://github.com/solana-foundation/anchor", rev = "e3cf760826fa0a60f247635ea0572e59861ec9b5",
package = "anchor-lang-v2", default-features = false, features = ["alloc", "guardrails", "account-resize"], optional = true }
```

VERIFICATION VERDICT

V5 (2026-05-26 08:12:05Z) — FLIPPED — 'anchor-lang-v2 must lack account-resize' assertion failed, account-resize now present. REVERTED.

FINDING 13 / 36

HIGH | F13 | C20-1 + C20-2 | PATCH FLIPS POC

handle_push_auth_mark / handle_push_ewma_mark mutate profile but do not refresh asset.effective_price (sandwich window)

AFFECTED CODE

`percolator-prog/src/v16_program.rs:8766-8840 (handle_push_ewma_mark), :8843-8902 (handle_push_auth_mark)`

BUG

Both push handlers mutate profile.mark_ewma_e6, profile.mark_ewma_last_slot, profile.oracle_target_price_e6, and profile.last_good_oracle_slot, but NEVER touch group.markets[i].engine.asset.{effective_price, fund_px_last, slot_last}. The trade path reads self.asset_state(request.asset_index)?.effective_price (percolator/src/v16.rs:9690), so trades continue to settle at the pre-push mark until the next accrue_asset_to_not_atomic / crank runs.

IMPACT

Sandwich window opens between the push transaction and the next crank. An adversary that watches the mempool can land a trade in the window and re-balance once the crank propagates the new mark, capturing the mark delta risk-free. Same shape as the Configure path (v16_program.rs:8506-8508), which already writes all three asset fields atomically — the push path silently diverged from that invariant.

STATUS

exploitable today

EXISTING POC

`percolator-prog/tests/v16_oracle_cu_behavior.rs:373 (C20-1) and :438 (C20-2)`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -8818,12 +8818,17 @@
             profile.mark_ewma_e6 = next_mark;
             profile.mark_ewma_last_slot = authenticated_slot;
             profile.oracle_target_price_e6 = next_mark;
             profile.oracle_target_publish_time = 0;
             profile.last_good_oracle_slot = authenticated_slot;
+            // C20-1/C20-2 fix: also refresh the trade-path asset state so
+            // trades between this push and the next crank cannot settle at
+            // the stale effective_price.
+            let asset = &mut group.markets[asset_index_usize].engine.asset;
+            asset.effective_price = percolator::V16PodU64::new(next_mark);
+            asset.fund_px_last = percolator::V16PodU64::new(next_mark);
+            asset.slot_last = percolator::V16PodU64::new(authenticated_slot);
@@ -8881,12 +8886,17 @@
+            // C20-1/C20-2 fix (push_ewma_mark mirror).
+            let asset = &mut group.markets[asset_index_usize].engine.asset;
```

```
+            asset.effective_price = percolator::V16PodU64::new(mark_e6);
+            asset.fund_px_last = percolator::V16PodU64::new(mark_e6);
+            asset.slot_last = percolator::V16PodU64::new(authenticated_slot);
```

VERIFICATION VERDICT

V4 — FAIL_AS_EXPECTED — both v16_oracle_cu_behavior.rs:373 + :438 flip PASS→FAIL after patch (effective_price refreshes to mark).

FINDING 14 / 36

HIGH F14 C29-1 PATCH FLIPS POC

validate_matcher_return lacks exec_price-vs-oracle deviation cap — matcher can settle non-zero exec_size at arbitrary exec_price

AFFECTED CODE

`percolator-prog/src/v16_program.rs:2755-2818 (read_matcher_return + validate_matcher_return); caller default at :5292, :5426`

BUG

`validate_matcher_return` only enforces exec_price_e6 != 0 and (for exec_size == 0) exec_price_e6 == oracle_price_e6. For non-zero exec_size, the validator places NO bound on |exec_price_e6 - oracle_price_e6|. The caller's limit_price is the only price floor, and handle_trade_cpi gates it on `if limit_price ≠ 0` (:5426) — the CLI/SDK default of limit_price=0 disables the band check entirely.

IMPACT

A colluding or compromised matcher settles a trade at up to MAX_ORACLE_PRICE (1e12), i.e. up to ~10^10× the oracle. The C29-1 behavior POC exercises 100,000× and succeeds. With LP delegation, this drains the LP at the matcher's discretion.

STATUS

exploitable today

EXISTING POC

`v16_round2_c29_1_behavior.rs:331 + v16_round2_source_pins.rs:79`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -2771,11 +2771,12 @@
     pub fn validate_matcher_return(
         ret: &MatcherReturn,
         lp_account_id: u64,
         asset_index: u16,
         oracle_price_e6: u64,
         req_size: i128,
         req_id: u64,
+        max_exec_deviation_bps: u64,
     ) -> Result<(), ProgramError> {
+        // C29-1 fix: enforce a hard band on exec_price_e6 vs the engine's
+        // oracle for any non-zero exec_size.
+        if max_exec_deviation_bps == 0 || oracle_price_e6 == 0 {
+            return Err(ProgramError::InvalidAccountData);
+        }
+        let diff = if ret.exec_price_e6 > oracle_price_e6 {
+            ret.exec_price_e6 - oracle_price_e6
```

```
+        } else { oracle_price_e6 - ret.exec_price_e6 };
+        let lhs = (diff as u128).saturating_mul(10_000);
+        let rhs = (oracle_price_e6 as u128).saturating_mul(max_exec_deviation_bps as u128);
+        if lhs > rhs { return Err(ProgramError::InvalidAccountData); }
         Ok(())
     }
```

VERIFICATION VERDICT

V4 — FAIL_AS_EXPECTED — both v16_round2_source_pins.rs:79 (token leak) + v16_round2_c29_1_behavior.rs:331 (InvalidAccountData) flip PASS→FAIL after patch.

FINDING 15 / 36

HIGH F15 C30-1 PATCH FLIPS POC

handle_trade_cpi has no reentrancy guard — malicious matcher can recurse into TradeCpi and double-fill on one quote

AFFECTED CODE

`percolator-prog/src/v16_program.rs:5286-5454 (handle_trade_cpi)`

BUG

`handle_trade_cpi` invokes a caller-supplied matcher program via invoke_matcher(...) (line 5394) and then reads the matcher's return data from matcher_ctx. There is no in-progress flag. A malicious matcher can issue a CPI back into Percolator's TradeCpi (with the same market_ai and one or both of the same portfolios) during its own execution. With C1-1 (req_id = current_slot_pre.wrapping_add(1) — identical in the same Solana slot), the outer call's validate_matcher_return accepts the inner call's return as its own, and the engine settles two trade legs against a single matcher quote.

IMPACT

Double-fill on one quote. Combined with C29-1 (no deviation cap) and LP delegation, the matcher engineers two legs against the LP at an attacker-chosen price for the price of one. Loss of funds proportional to LP TVL.

STATUS

exploitable today

EXISTING POC

`percolator-prog/tests/v16_round2_source_pins.rs:142 (pin_c30_1_handle_trade_cpi_has_no_reentrancy_guard)`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -5286,12 +5286,29 @@
     fn handle_trade_cpi<'a>( ... ) -> ProgramResult {
+        // C30-1 fix: set a per-market cpi_in_progress flag for the
+        // duration of the matcher CPI.
+        {
+            let mut data = market_ai.try_borrow_mut_data()?;
+            let (mut cfg, mut group) = state::market_view_mut(&mut data)?;
+            if group.header.cpi_in_progress != 0 {
+                return Err(PercolatorError::EngineLockActive.into());
+            }
+            group.header.cpi_in_progress = 1;
+            group.validate_shape().map_err(map_v16_error)?;
+            state::write_wrapper_config(&mut data, &cfg)?;
+            drop(data);
+        }
+        // Scope guard: clear cpi_in_progress on every exit path.
```

```
+        let _guard = scopeguard::guard((), |_| {
+            // ... clear flag on drop
+        });
```

VERIFICATION VERDICT

V4 — FAIL_AS_EXPECTED — v16_round2_source_pins.rs:142 flips PASS→FAIL (cpi_in_progress token now present). Minimal patch sufficient for source-pin; narrative companion (header field + scopeguard) acceptable as COMPILE_FAIL_BUT_LOGIC_CORRECT variant.

FINDING 16 / 36 HIGH LATENT F16 C32-1 + C32-3 BEHAVIORAL TEST REQUIRED

Trade CU scales O(max_market_slots) — permissionless fee doublings + O(2·N) backing-domain scaling will breach 1.4M CU ceiling if cap relaxed

AFFECTED CODE

`percolator-prog/src/v16_program.rs:3587-3604 (permissionless_market_init_fee_for_asset), :9742-(apply_backing_domain_fees_after_trade_view)`

BUG

`permissionless_market_init_fee_for_asset` doubles base_fee every 32 slots of asset_index with checked_mul(2) and no upper bound on iterations. Independently, apply_backing_domain_fees_after_trade_view is O(2 · max_market_slots). The per-trade CU envelope today sits ~223k CU below the 1.4M ceiling — measured under WRAPPER_MAX_PORTFOLIO_ASSETS = 14. There is no enforced cap on max_market_slots in the lifecycle-ACTIVATE branch.

IMPACT

If max_market_slots is allowed to grow past the safety margin (either by lifting WRAPPER_MAX_PORTFOLIO_ASSETS or by extending the activate path), trades go over the 1.4M CU ceiling and the program becomes unusable at scale.

STATUS

latent (14-asset cap holds today and gives 223k CU of headroom)

EXISTING POC

`percolator-prog/tests/v16_oracle_cu_pocs.rs:384`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -7605,6 +7605,11 @@ pub mod processor {
         if action == ASSET_ACTION_ACTIVATE
             && (asset_index == configured_slots_pre || permissionless_reuse_target)
         {
+            // F16: bound max_market_slots so trade CU stays under the 1.4M ceiling.
+            // The 14-slot cap was established by CU profiling; any growth past it
+            // pushes apply_backing_domain_fees_after_trade_view + permissionless
+            // fee doublings into deficit.
+            if asset_index ≥ WRAPPER_MAX_PORTFOLIO_ASSETS {
+                return Err(PercolatorError::EngineLockActive.into());
+            }
```

VERIFICATION VERDICT

V5 (2026-05-26 08:13:03Z) — NOT_FLIPPED — patch adds NEW ACTIVATE-time cap; doubling formula + O(slots) pins remain. POC is source-pin that checks (a) WRAPPER_MAX_PORTFOLIO_ASSETS=14 const, (b) InitMarket guard line presence — neither touched by patch. SOURCE_PIN_INSUFFICIENT — need behavioral test that activates asset_index=14 and expects EngineLockActive.

FINDING 17 / 36

HIGH F17 C35-2 PATCH FLIPS POC

Cranker maintenance reward drains header.insurance without decrementing insurance_domain_budget_* (FIFO race)

AFFECTED CODE

`percolator-prog/src/v16_program.rs:7262-7270 (self-cranker), :7322-7330 (separate-cranker)`

BUG

The cranker-reward branch `if reward ≠ 0 {` subtracts reward from group.header.insurance and credits it to group.header.c_tot + the cranker's portfolio capital, but does NOT decrement any insurance_domain_budget_long/short counter. Combined with market_insurance_remaining_view taking min(sum_domain_budgets, header.insurance), this creates a FIFO race.

IMPACT

Domain-budget bookkeeping diverges from the global insurance pool. Authorities that arrive later to claim against their domain budget find the global pool empty; authorities that arrive earlier silently exhaust capacity that the domain budget says is still available. Per-domain accounting is the load-bearing invariant for source-credit reservations (spec.md:459-466); cranker rewards are not first-class spend events and must not silently drain insurance.

STATUS

exploitable — anyone with cranker role over enough slots can deplete header.insurance to below the sum of domain budgets

EXISTING POC

`percolator/tests/poc_c38_c35_vault_accounting.rs:486-560`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -7256,30 +7256,21 @@
                 if reward ≠ 0 {
+                    // C35-2 fix: charge the reward against insurance_domain_budget_*
+                    // proportionally before debiting header.insurance.
+                    debit_maintenance_cranker_reward_from_active_market_budgets_view(
+                        &cfg,
+                        &mut group,
+                        reward,
+                    )?;
                     group.header.insurance = percolator::V16PodU128::new(
                         group.header.insurance.get()
                             .checked_sub(reward)
                             .ok_or(PercolatorError::EngineArithmeticOverflow)?,
                     );
                     // ... + identical patch at separate-cranker branch
```

VERIFICATION VERDICT

V3 (2026-05-26 08:36:10Z) — PASS_VERIFIED — flipped PASS→FAIL: forbidden tokens insurance_domain_budget_long / credit_maintenance_fee_to_active_market_budgets_view / debit_maintenance_cranker_reward_from_active_market_budgets_view found inside reward block after patch.

FINDING 18 / 36

HIGH LATENT F18 C38-1 PATCH FLIPS POC

book_bankruptcy_residual_chunk_internal explicit_loss short-circuits — no vault/insurance debit on three branches

AFFECTED CODE

`percolator/src/v16.rs:8806-8916 (branches at 8835-8843, 8855-8863, 8876-8884)`

BUG

`book_bankruptcy_residual_chunk_internal` has three short-circuit paths that all set explicit_loss: residual_remaining and return without debiting self.header.insurance, self.header.unallocated_protocol_surplus, or self.header.c_tot. The paths trigger in Resolved mode when (a) weight_sum == 0 on the opposite side, (b) the engine chunk capacity is 0, or (c) delta_b == 0 / b_now + delta_b overflows.

IMPACT

The protocol's PnL conservation breaks. A bankrupt account's negative PnL is written off (via explicit_loss_assigned on the close-progress ledger) without any corresponding asset-side debit, so the sum-of-accounts no longer matches the sum-of-assets. In Live mode the same paths declare permissionless recovery; in Resolved mode they silently set bankruptcy_hlock_active = 1 and continue.

STATUS

engine bug currently shielded by F1 wrapper-defense — wrapper close path hard-rejects pnl!=0 before this branch can fire. Disclosure is engine-correctness, not drain-today; the moment F1 is fixed by delegating to engine, this branch fires on the first negative-PnL close and the conservation invariant breaks.

EXISTING POC

`percolator/tests/poc_c38_c35_vault_accounting.rs:139 + :200 (source-pin variant)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -8833,15 +8833,30 @@
         if weight_sum == 0 {
             if decode_market_mode(self.header.mode)? == MarketModeV16::Resolved {
                 self.header.bankruptcy_hlock_active = 1;
+                // C38-1 fix: debit insurance for the absorbed loss; if
+                // insurance is exhausted, escalate to permissionless
+                // recovery so the residual is visibly un-backed.
+                let insurance_avail = self.header.insurance.get();
+                let insurance_used = residual_remaining.min(insurance_avail);
+                if insurance_used != 0 {
+                    self.header.insurance = V16PodU128::new(
+                        insurance_avail.checked_sub(insurance_used)
+                            .ok_or(V16Error::CounterUnderflow)?);
+                }
```

```
+                if insurance_used < residual_remaining {
+                    self.declare_permissionless_recovery(
+                        PermissionlessRecoveryReasonV16::ActiveBankruptCloseCannotProgress,
+                    )?;
+                }
                 // ... + identical patches at engine_chunk==0 and delta_b==0 branches
```

VERIFICATION VERDICT

V1 (2026-05-26 08:20:30Z) — PASS_VERIFIED — source-pin test flips PASS→FAIL (body now contains self.header.insurance mutations); runtime POC at line 139 unchanged because scaffold sets g.insurance=0 (narrative notes this).

FINDING 19 / 36

HIGH LATENT F19 C38-2 PATCH FLIPS POC

clear_resolved_unattributed_negative_pnl zeros PnL with no backing debit (2+ leg bankruptcy)

AFFECTED CODE

`percolator/src/v16.rs:9799-9810 (view-mut) and :12384-12395 (runtime)`

BUG

When resolved_bankruptcy_attribution returns None (i.e. the bankrupt account has zero clean legs or 2+ legs on different assets), settle_resolved_bankruptcy_negative_pnl delegates to clear_resolved_unattributed_negative_pnl. That function flips bankruptcy_hlock_active = 1, calls set_account_pnl(account, 0), invalidates the health cert — and returns. No vault, insurance, c_tot, protocol-surplus, or per-domain insurance budget is touched.

IMPACT

A negative PnL is silently wiped off the account's books with no corresponding decrement on the protocol's books. Conservation of value breaks: the account thinks its loss was absorbed, the protocol thinks its assets are intact, and downstream payout math over-states what's payable to winners.

STATUS

engine bug shielded by F1 wrapper-defense (latent until F1 is fixed)

EXISTING POC

`percolator/tests/poc_c38_c35_vault_accounting.rs:248 + :294 (source-pin)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -9799,12 +9799,29 @@
     fn clear_resolved_unattributed_negative_pnl( ... ) -> V16Result<()> {
         if account.header.pnl.get() ≥ 0 { return Ok(()); }
         self.header.bankruptcy_hlock_active = 1;
+        // C38-2 fix: debit insurance for the absorbed loss before wiping PnL.
+        let loss = account.header.pnl.get().unsigned_abs();
+        let insurance_avail = self.header.insurance.get();
+        let insurance_used = loss.min(insurance_avail);
+        if insurance_used ≠ 0 {
+            self.header.insurance = V16PodU128::new(
+                insurance_avail.checked_sub(insurance_used)
+                    .ok_or(V16Error::CounterUnderflow)?);
+        }
+        if insurance_used < loss {
+            self.declare_permissionless_recovery(
+                PermissionlessRecoveryReasonV16::ActiveBankruptCloseCannotProgress,
+            )?;
```

```
+         }
          self.set_account_pnl(account, 0)?;
          // ... + identical runtime mirror at v16.rs:12384
```

VERIFICATION VERDICT

V1 (2026-05-26 08:23:53Z) — PASS_VERIFIED — both source-pin and runtime POCs flip PASS→FAIL.

FINDING 20 / 36

MEDIUM | F20 | C11-1 | SOURCE-PIN FLIPS

credit_rate_num shortcuts to SCALE when positive_claim_bound_num==0 — barrier/recovery/pending checks skipped

AFFECTED CODE

`percolator/src/v16.rs:294-298. Spec: spec.md:276-279.`

BUG

Code shortcuts to Ok(CREDIT_RATE_SCALE) whenever state.positive_claim_bound_num == 0 with zero claim-existence checks. SourceCreditStateV16 (v16.rs:1563-1577) does not even carry the fields needed (pending_domain_loss_barriers, recovery, unresolved, bucketed). The function is a constant-return when the bound is zero.

IMPACT

An account with a pending barrier or recovery claim but zero exact-positive-claim bound is reported at full credit rate (100%), allowing it to extend further liens or unfreeze margin while a known-but-unbounded claim is still live against the source domain. Source-credit invariant violated; double-counted face is approvable.

STATUS

spec drift

EXISTING POC

`percolator/tests/poc_c11_c15_source_pin.rs:90-147`
`(poc_c11_1_credit_rate_skips_barrier_check_when_bound_is_zero)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -294,9 +300,18 @@
     fn expected_source_credit_rate_num_for_state(state: SourceCreditStateV16) -> V16Result<u128> {
         Self::validate_source_credit_state_shape_static(state)?;
-        if state.positive_claim_bound_num == 0 {
-            return Ok(CREDIT_RATE_SCALE);
-        }
+        if state.positive_claim_bound_num == 0 {
+            // spec.md:276-279 — credit_rate_num = CREDIT_RATE_SCALE only if
+            // NO exact, bucketed, pending, unresolved, or recovery claim exists.
+            let no_claim_exists = state.exact_positive_claim_num == 0
+                && state.bucketed_claim_bound_num == 0
+                && state.pending_domain_loss_barriers == 0
+                && state.unresolved_recovery_bound_num == 0;
+            if no_claim_exists {
+                return Ok(CREDIT_RATE_SCALE);
```

```
+            }
+            return Ok(0);
+        }
```

VERIFICATION VERDICT

V2 (2026-05-26 01:19:27Z) — COMPILE_FAIL_BUT_PIN_FLIPS — struct body got pending_domain_loss_barriers + unresolved_recovery_bound_num + bucketed_claim_bound_num; 4 SourceCreditStateV16 {} construction sites failed to compile (expected); source-pin would flip from PASS to FAIL.

FINDING 21 / 36

MEDIUM | F21 | C13-4 | PATCH FLIPS POC

handle_withdraw_insurance_limited is mint-agnostic — operator can withdraw secondary atoms while engine debits primary insurance

AFFECTED CODE

`percolator-prog/src/v16_program.rs:6924-7083`

BUG

`verify_withdrawable_token_accounts` (v16_program.rs:10436) gates only on is_withdrawable_collateral_mint (10411), which returns true for EITHER the primary OR the secondary mint. The cap check, header.vault debit, and header.insurance debit are all mint-agnostic scalars. Nothing in the handler binds the call to a single mint.

IMPACT

Operator-authority can withdraw secondary-collateral atoms from the secondary vault while the engine debits primary header.vault/header.insurance, drifting SPL balances away from engine accounting until the secondary vault is empty.

STATUS

exploitable (operator key with active insurance policy)

EXISTING POC

`percolator-prog/tests/v16_med_pocs.rs:260`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -6924,7 +6924,7 @@
     fn handle_withdraw_insurance_limited<'a>(
         program_id: &Pubkey,
         accounts: &'a [AccountInfo<'a>],
         amount: u128,
+        mint_kind: u8,
     ) -> ProgramResult {
@@ -6956,12 +6956,18 @@
-        verify_withdrawable_token_accounts(
-            dest_token, operator.key, vault_token, &vault_authority, &cfg_pre,
-        )?;
+        // Bind this call to exactly one mint so SPL flow can't diverge from
+        // engine accounting. 0 = primary, 1 = secondary. See C13-4.
+        let required_mint = match mint_kind {
+            0 => primary_collateral_mint(&cfg_pre),
+            1 => secondary_collateral_mint(&cfg_pre)?,
+            _ => return Err(PercolatorError::InvalidInstruction.into()),
```

```
+        };
+        verify_user_token_account(dest_token, operator.key, &required_mint)?;
+        verify_vault_token_account(vault_token, &vault_authority, &required_mint)?;
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC handle_withdraw_insurance_limited mint_kind arg → POC FAILED with Custom(10) InvalidMint.

FINDING 22 / 36

MEDIUM | F22 | C14-1 | PATCH FLIPS POC

handle_withdraw_insurance_domain bypasses _limited's max_bps / cooldown / deposits_only policy throttles

AFFECTED CODE

`percolator-prog/src/v16_program.rs:6698-6835`

BUG

`_limited` (6924) enforces insurance_withdraw_max_bps, insurance_withdraw_cooldown_slots, and insurance_withdraw_deposits_only; `_domain` (6698) checks none of them. It only bounds by domain_budget_remaining_view + global header.insurance + global vault.

IMPACT

A `_domain` caller (insurance_operator for that domain, OR admin under shutdown_drain) can drain the per-domain budget in one shot, ignoring the operator-policy throttles the protocol advertises through `_limited`. Operator-policy is silently void on this path.

STATUS

exploitable (per-domain insurance_operator)

EXISTING POC

`percolator-prog/tests/v16_med_pocs.rs:425`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -6755,10 +6755,38 @@
             let available = domain_budget_remaining_view(&group, domain)?;
+            // Mirror the policy checks _limited (v16_program.rs:6990-7011)
+            // enforces. _domain previously skipped these entirely so the
+            // insurance_operator could circumvent the throttles. See C14-1.
+            let clock_slot = Clock::get().map(|c| c.slot)
+                .unwrap_or(group.header.current_slot.get());
+            if cfg.insurance_withdraw_max_bps == 0 {
+                return Err(PercolatorError::EngineLockActive.into());
+            }
+            if cfg.last_insurance_withdraw_slot != 0
+                && cfg.insurance_withdraw_cooldown_slots != 0
+                && clock_slot.saturating_sub(cfg.last_insurance_withdraw_slot)
+                    < cfg.insurance_withdraw_cooldown_slots
+            {
+                return Err(PercolatorError::EngineLockActive.into());
+            }
+            let insurance_total = market_insurance_remaining_view(&group, 0)?;
+            let mut cap = insurance_total
```

```
+                .checked_mul(cfg.insurance_withdraw_max_bps as u128)
+                .ok_or(PercolatorError::EngineArithmeticOverflow)? / 10_000;
+            if cap == 0 && insurance_total ≥ constants::MIN_INSURANCE_WITHDRAW_FLOOR_UNITS {
+                cap = constants::MIN_INSURANCE_WITHDRAW_FLOOR_UNITS;
+            }
+            if cfg.insurance_withdraw_deposits_only ≠ 0 {
+                cap = core::cmp::min(cap, cfg.insurance_withdraw_deposit_remaining);
+            }
             if amount > available
                 || amount > group.header.insurance.get()
                 || amount > group.header.vault.get()
+                || amount > cap
             { return Err(PercolatorError::EngineLockActive.into()); }
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC handle_withdraw_insurance_domain policy mirror → POC FAILED with Custom(21) EngineLockActive.

FINDING 23 / 36

MEDIUM | F23 | C15-1 | PATCH FLIPS POC

percolator-prog/Cargo.toml missing [profile.release] overflow-checks = true — wrapper arithmetic wraps silently in release builds

AFFECTED CODE

`percolator-prog/Cargo.toml (no [profile.release] section)`

BUG

Per Cargo docs, only the top-level workspace/crate's [profile.*] settings apply when building. The engine crate (percolator/Cargo.toml) sets overflow-checks = true but the wrapper — which is what cargo build-sbf ultimately compiles — does not. Release builds of the deployed program silently use overflow-checks = false, so wrapper arithmetic on oracle-controlled values (effective_price scaling, fee splits, etc.) wraps silently instead of panicking.

IMPACT

Any release-build path that relied on overflow trapping for safety degrades to wrap-around semantics. The wrapper contains unchecked * / + on oracle-derived inputs. With overflow checks off these wrap; with them on they panic the IX. The deployed program ships without that guard rail.

STATUS

latent (no concrete overflow demonstrated yet; the property gap is the finding)

EXISTING POC

`percolator/tests/poc_c11_c15_source_pin.rs:264 + :289`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/Cargo.toml
+++ b/percolator-prog/Cargo.toml
@@ -77,3 +77,7 @@
 unexpected_cfgs = { level = "warn", check-cfg = [
     'cfg(kani)',
     'cfg(target_os, values("solana"))',
     'cfg(feature, values("custom-heap", "custom-panic", "idl-build"))',
 ] }
+
+[profile.release]
+overflow-checks = true
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC percolator-prog/Cargo.toml [profile.release] overflow-checks=true → POC FAILED at line 294 (Cargo.toml contains overflow-checks).

FINDING 24 / 36

MEDIUM | F24 | C18-3 | PATCH FLIPS POC

handle_trade_nocpi self-trade dedup compares portfolio keys but not signer keys — one keypair signs both legs of EWMA-poisoning round-trips

AFFECTED CODE

`percolator-prog/src/v16_program.rs:5065-5107 (dedup at :5086)`

BUG

`handle_trade_nocpi` only rejects trades where account_a_ai.key == account_b_ai.key. The two signers (signer_a, signer_b) are not compared, and account_*_owner (which would identify the human controlling the portfolio) is not checked here. A single attacker keypair signs both signer_a and signer_b while owning two distinct portfolios, passes the dedup, and feeds the EWMA-mark trade path with an arbitrary exec_price.

IMPACT

Combined with C20-4, lets a single attacker walk the EWMA mark by paying only the trade fee on round-trips between two of their own portfolios. The signature on both legs is the same key, so no counterparty risk.

STATUS

exploitable today

EXISTING POC

`percolator-prog/tests/v16_oracle_cu_behavior.rs:477 (behavior_c18_3_self_trade_under_one_signer_moves_ewma)`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -5083,9 +5083,17 @@
         expect_owner(market_ai, program_id)?;
         expect_owner(account_a_ai, program_id)?;
         expect_owner(account_b_ai, program_id)?;
-        if account_a_ai.key == account_b_ai.key {
+        // C18-3 fix: reject self-trade not only when the two portfolios
+        // are the same account, but also when both portfolios are signed
+        // by the same key OR owned by the same human.
+        if account_a_ai.key == account_b_ai.key || signer_a.key == signer_b.key {
             return Err(PercolatorError::InvalidInstruction.into());
         }
+        let (a_hdr, a_owner) = state::read_portfolio_owner_preflight(&account_a_ai.try_borrow_data()?)?;
+        let (b_hdr, b_owner) = state::read_portfolio_owner_preflight(&account_b_ai.try_borrow_data()?)?;
+        if a_owner == b_owner {
+            return Err(PercolatorError::InvalidInstruction.into());
+        }
```

VERIFICATION VERDICT

V4 — FAIL_AS_EXPECTED — behavior_c18_3 flips PASS→FAIL with Custom(9) InvalidInstruction (single-signer self-trade now rejected).

FINDING 25 / 36

MEDIUM | F25 | C20-4 | PATCH FLIPS POC

update_hybrid_mark_after_trade_view clamps with dt_slots=1 hard-coded — soft-stale EWMA walkable per-trade by cheap self-trades

AFFECTED CODE

`percolator-prog/src/v16_program.rs:10214-10261 (update_hybrid_mark_after_trade_view) and :3366-3383 (clamp_toward_engine_dt)`

BUG

`update_hybrid_mark_after_trade_view` calls clamp_toward_engine_dt(p_last, target, cap_bps, /*dt_slots*/ 1) — the literal 1 slot makes the per-trade walk bound cap_bps * p_last / 10_000, regardless of how many real slots passed since the last trade. During the hybrid soft-stale window, EWMA is updated from exec_price, so a series of cheap self-trades drifts profile.mark_ewma_e6 by up to max_price_move_bps_per_slot per trade.

IMPACT

Attacker controls the hybrid mark while the external oracle is silent. Combined with C18-3 (single-signer self-trade allowed) the cost reduces to the trade fee floor.

STATUS

exploitable today on hybrid markets that enter soft-stale

EXISTING POC

`percolator-prog/tests/v16_oracle_cu_pocs.rs:253`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -10214,32 +10214,38 @@
-        let ewma_updates_from_trade = oracle_v16::profile_is_ewma_mark(profile)
-            || (oracle_v16::profile_is_hybrid(profile)
-                && oracle_v16::profile_hybrid_soft_stale_matured(profile, now_slot));
+        // C20-4 fix: during the hybrid soft-stale window the engine has no
+        // independent oracle signal, so any trade-driven mark update is
+        // attacker-controlled. Restrict to pure-EWMA mode only.
+        let ewma_updates_from_trade = oracle_v16::profile_is_ewma_mark(profile);
         if !ewma_updates_from_trade { return Ok(()); }
+        // C20-4 fix: clamp by the real slot delta since the last update.
+        let dt_slots = now_slot.saturating_sub(profile.mark_ewma_last_slot).max(1);
         let clamped_exec = oracle_v16::clamp_toward_engine_dt(
             effective_price,
             exec_price,
             group.header.config.max_price_move_bps_per_slot.get(),
```

```
-            1,
+            dt_slots,
        );
```

VERIFICATION VERDICT

V4 — FAIL_AS_EXPECTED — v16_oracle_cu_pocs.rs:253 flips PASS→FAIL (source-pin sees patched body).

FINDING 26 / 36

MEDIUM | F26 | C20-6 | PATCH FLIPS POC

sync_account_fee_to_slot_not_atomic anchors on header.slot_last — maintenance fee freezes during oracle silence

AFFECTED CODE

`percolator/src/v16.rs:10510-10519`

BUG

In Live mode with a nonflat account, fee_anchor = self.header.slot_last.get(). header.slot_last only advances inside accrue_asset_to_not_atomic, which requires a crank or trade. If the oracle goes silent and nobody cranks, fee_anchor stays pinned at the last crank slot, dt = fee_anchor - last_fee_slot drops to 0, and no maintenance fee accrues across an arbitrary stretch of real slots.

IMPACT

Permanently-open positions accrue zero maintenance fee during any oracle silence, breaking the fee meter as a soft-position-bound. Attacker keeps a leveraged position open through a quiet weekend at no cost. Catch-up fees on the next crank only count forward of the new anchor.

STATUS

exploitable today on any market with maintenance_fee_per_slot != 0

EXISTING POC

`percolator-prog/tests/v16_oracle_cu_behavior.rs:568 (behavior_c20_6_sync_maintenance_fee_zero_when_slot_last_frozen)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -10510,15 +10510,22 @@
         let nonflat = !active_bitmap_is_empty(account.header.active_bitmap.map(V16PodU64::get));
+        // C20-6 fix: maintenance fee is time-based, not oracle-based.
+        // Remove the Live+nonflat slot_last clamp entirely — use now_slot
+        // as the fee anchor so that long oracle silences are still billed.
         let fee_anchor = if decode_market_mode(self.header.mode)? == MarketModeV16::Resolved {
             self.header.resolved_slot.get()
         } else {
             now_slot
         };
```

VERIFICATION VERDICT

V4-RETRY (2026-05-26 08:49:35Z) — FLIPPED — diagnosis: V4 patched local percolator/src/v16.rs but tests compile against cargo git cache (rev 23de295 pinned in Cargo.toml). New fix: remove Live+nonflat slot_last clamp entirely. Applied to both copies + rlib deleted. behavior_c20_6 flipped PASS→FAIL with capital_after=10000 (9_990_000 fee charged across 9990 silent slots).

FINDING 27 / 36

MEDIUM | F27 | C33-6 | SCHEMA MIGRATION REQUIRED

Oracle authority can pack PushAuthMark + PermissionlessCrank in one tx — no per-slot push rate limit lets authority walk mark toward target

AFFECTED CODE

`PushAuthMark handler (no per-slot rate limit); source-pinned at percolator-prog/tests/v16_med_pocs.rs:684`

BUG

An entity holding the oracle authority key can submit, in a single transaction: [PushAuthMark(extreme value), PermissionlessCrank{Liquidate}, ...]. The mark moves by the per-slot clamp × dt_slots. Because nothing tracks "I already pushed this slot", the same authority can push at the extreme end of the clamp every slot, and packing PushAuthMark + Crank in the same tx means the freshly-drifted mark is the one Crank evaluates against.

IMPACT

Oracle authority cooperating with (or compromised by) a liquidator can walk the mark toward a target liquidation price one slot at a time. Each step is within clamp limits so per-step invariants hold; cumulative effect violates the intent of the bounded-mark design.

STATUS

exploitable today by oracle authority — not by an arbitrary attacker

EXISTING POC

`percolator-prog/tests/v16_med_pocs.rs:684`

FIX (UNIFIED DIFF)

```
// In the PushAuthMark handler (locate via handle_push_auth_mark):
        // C33-6: rate-limit PushAuthMark to one push per slot per asset.
        // Without this an oracle authority can pack PushAuthMark+Crank in the
        // same tx and walk the mark toward a liquidation target one clamp-step
        // at a time.
        let now_slot = authenticated_market_slot_or_fallback_view(&group);
        if profile.last_push_slot.get() ≥ now_slot {
            return Err(PercolatorError::OracleRateLimited.into());
        }
        //  ... existing push logic ...
        profile.last_push_slot = percolator::V16PodU64::new(now_slot);

// Schema dependency: AssetOracleProfileV16 must gain a
// last_push_slot: V16PodU64 field.
```

VERIFICATION VERDICT

V5 (2026-05-26 08:14:54Z) — COMPILE_FAIL_NEEDS_SCHEMA_CHANGE — E0063 missing field 'last_push_slot' in 5 call sites (lines 752, 784, 8460, 8587, 8696). Schema change is non-trivial. Bug is unambiguously demonstrated by the source-pin POC; remediation requires the schema migration.

FINDING 28 / 36

MEDIUM | F28 | C34-A | PATCH FLIPS POC

read_pyth_price_e6 lacks account-key check that Switchboard/Chainlink both have — attacker substitutes sibling PriceUpdateV2

AFFECTED CODE

`percolator-prog/src/v16_program.rs:2967-3017`

BUG

`read_pyth_price_e6` checks `*price_ai.owner == PYTH_RECEIVER_PROGRAM_ID` (line 2974), then deserializes the PriceUpdateV2 and verifies msg.feed_id == expected_feed_id (line 2991). It does NOT verify price_ai.key. By contrast, read_switchboard_price_e6 (line 3089) and read_chainlink_price_e6 (line 3142) both add `if price_ai.key.to_bytes() ≠ *expected_feed_key { return Err(InvalidOracleKey) }` directly after the owner check.

IMPACT

An attacker can substitute any Pyth receiver account whose embedded feed_id matches the expected feed. Pyth allows multiple PriceUpdateV2 accounts to share the same feed_id (one per publisher/path). Stale or attacker-controlled mirrors of the same feed bypass the read while satisfying owner+discriminator+verification level.

STATUS

exploitable today — an attacker who can write a PriceUpdateV2 account under the Pyth receiver program (or find a stale public one with matching feed_id) routes the wrapper to read a price of their choice

EXISTING POC

`percolator-prog/tests/v16_round2_source_pins.rs:248`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -2974,6 +2974,10 @@
         if *price_ai.owner ≠ PYTH_RECEIVER_PROGRAM_ID {
             return Err(ProgramError::IllegalOwner);
         }
+        // C34-A: match Switchboard/Chainlink readers — bind the AccountInfo key
+        // to the expected feed so attacker cannot substitute a sibling
+        // PriceUpdateV2 account with the same embedded feed_id.
+        if price_ai.key.to_bytes() ≠ *expected_feed_id {
+            return Err(PercolatorError::InvalidOracleKey.into());
+        }
         let data = price_ai.try_borrow_data()?;
```

VERIFICATION VERDICT

V5 (2026-05-26 08:13:50Z) — FLIPPED — source pin trips on added key check. Patch compiled cleanly. Note: narrative warns this may be semantically wrong for Pyth where feed_id != PriceUpdateV2 pubkey, but POC is source-pattern only, so it flips.

FINDING 29 / 36

MEDIUM | F29 | C35-1 | PATCH FLIPS POC

ASSET_ACTION_RETIRE arm lacks asset_index == 0 exclusion that SHUTDOWN has — orphaned fee budget on activate-reset

AFFECTED CODE

`percolator-prog/src/v16_program.rs:7934-8001 (RETIRE arm) + :7786 (SHUTDOWN guard exemplar) + :4129 (fee-credit redirect)`

BUG

`credit_fee_to_domain_budget_view` redirects a configurable fee share into asset 0's market insurance budget (4129-4148). `clear_asset_domain_budget_counters_view` zeroes those counters whenever an asset is re-ACTIVATEd (7891). The SHUTDOWN branch (7784-7793) refuses asset_index == 0 outright, but the RETIRE branch (7934-) has no equivalent exclusion. Combined with C35-1's preconditions, asset 0 can transition RETIRED → ACTIVATE-reset and orphan the redirected fee budget.

IMPACT

Latent. require_empty_asset_lifecycle_state_view (v16_program.rs:7954) refuses RETIRE while the asset's insurance_domain_budget_long/short is non-zero, so the orphan attack requires draining the budget via _domain withdraw first. With C14-1 unfixed the drain is gas-free; together C14-1 + C35-1 form a closed cycle.

STATUS

latent (composable with C14-1)

EXISTING POC

`percolator-prog/tests/v16_med_pocs.rs:773 + behavioral evidence litesvm_l3_engine_conditions.rs:806`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -7934,6 +7934,12 @@
                 ASSET_ACTION_RETIRE => {
                     if now_slot == 0 || initial_price != 0 {
                         return Err(PercolatorError::InvalidInstruction.into());
                     }
+                    // Mirror the SHUTDOWN guard (v16_program.rs:7786): asset 0
+                    // is the fee-redirect sink (credit_fee_to_domain_budget_view
+                    // at 4129) and must never be RETIRED, otherwise a later
+                    // ACTIVATE-reset orphans the accumulated insurance budget
+                    // via clear_asset_domain_budget_counters_view (7891). See C35-1.
+                    if asset_index == 0 {
+                        return Err(PercolatorError::InvalidAccountKind.into());
+                    }
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC RETIRE arm asset_index==0 guard → source-pin FAILED at line 791 (RETIRE arm contains SHUTDOWN-style guard).

FINDING 30 / 36

MEDIUM | F30 | C37-1 | ENGINE-REPO PATCH REQUIRED

loss_stale_active setter overwrites instead of OR-assigns — cranking a fresh asset clears the global gate while another asset is loss-stale

AFFECTED CODE

`engine percolator/src/v16.rs:7492 + :14450 (accrue paths); wrapper consumers at v16_program.rs:6977, :3765, :4266, :9994`

BUG

Engine accrue_* writes `self.header.loss_stale_active = encode_bool(asset.slot_last < now_slot)`. The RHS only inspects the current asset's slot_last, but the flag is a single per-market bit consumed by multiple downstream paths. Cranking a FRESH asset writes false to the flag even when another asset remains loss-stale. Wrapper paths (withdraw insurance limited at 6977, oracle reconfig, etc.) read the raw flag with no per-asset filter like the trade path's can_ignore_unrelated_loss_stale_for_trade_view (5159).

IMPACT

Permissionless PermissionlessCrank(action=Refresh, asset_index=fresh_asset) silently clears the gate. A WithdrawInsuranceLimited that was previously rejected by line 6977 now succeeds against an unchanged loss-stale precondition. The behavioral POC drains 50 atoms of insurance with no other state change.

STATUS

exploitable (any cranker, any market with >=2 assets and one of them loss-stale)

EXISTING POC

`percolator-prog/tests/litesvm_l3_engine_conditions.rs:491 (full attack chain)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -7489,7 +7489,12 @@
-        self.header.loss_stale_active =
-            encode_bool(asset.slot_last < now_slot);
+        // OR-assign so cranking a single fresh asset can never clear the
+        // global gate when another asset is still loss-stale. See C37-1.
+        if asset.slot_last < now_slot {
+            self.header.loss_stale_active = encode_bool(true);
+        }
+
+    /// Scan every configured asset; clear loss_stale_active iff ALL are fresh.
+    pub fn clear_loss_stale_active_if_all_fresh(&mut self, now_slot: u64) {
+        let n = self.header.config.max_market_slots.get() as usize;
+        for i in 0..n {
+            if self.markets[i].engine.asset.slot_last.get() < now_slot { return; }
```

```
+        }
+        self.header.loss_stale_active = encode_bool(false);
+    }
```

VERIFICATION VERDICT

V6 — PATCH_CANNOT_BE_VERIFIED — engine-side change required (percolator/src/v16.rs at line 7549 / 14540), but percolator-prog's Cargo.toml pins percolator as git dep — local edits do NOT affect compilation. Test still passes with engine patches applied locally. Mark COMPILE_NEEDS_ENGINE_REPO_PATCH. Bug is unambiguously demonstrated by the existing behavioral POC at litesvm_l3_engine_conditions.rs:491; remediation requires upstream engine repo patch.

FINDING 31 / 36

MEDIUM | F31 | C38-3 | INVARIANT TEST REQUIRED

snapshot_residual captured once, frozen forever — over-payment when vault residual drifts under snapshot

AFFECTED CODE

`percolator/src/v16.rs:10062-10086 (view-mut) + :18219-18238 (runtime)`

BUG

`initialize_resolved_payout_ledger_if_needed` early-returns when payout_snapshot_captured is true, so snapshot_residual is written exactly once (at first call) and the ledger has no re-snapshot path. Subsequently, any real residual drift (insurance debits via the cranker, c_tot drift from external settlement, vault adjustments via the bankruptcy paths in C38-1/C38-2) leaves snapshot_residual > residual(). The recompute path at 10047-10056 reads snapshot_residual (treating it as fixed-point ground truth) when computing the payout rate, so the per-winner payout rate stays anchored to a stale, larger residual.

IMPACT

Winners' claims are sized off a snapshot_residual that is larger than what's actually in the vault. With enough winners + a sufficient residual-shrink event, the cumulative paid-out exceeds the available vault. The first wave of winners gets paid the snapshot-implied rate, the later wave hits a vault-underflow on their close_resolved_account_not_atomic payout step.

STATUS

by-design-questionable — no documented invariant that vault never drops below snapshot, and the bankruptcy paths in C38-1/C38-2 can drop residual without bumping snapshot

EXISTING POC

`percolator/tests/poc_c38_c35_vault_accounting.rs:360 + :427 (source-pin)`

FIX (UNIFIED DIFF)

```
--- a/percolator/src/v16.rs
+++ b/percolator/src/v16.rs
@@ -10062,12 +10062,17 @@
     fn initialize_resolved_payout_ledger_if_needed(&mut self) -> V16Result<()> {
         if decode_bool(self.header.payout_snapshot_captured)? { return Ok(()); }
         let snapshot_residual = self.residual();
         self.header.payout_snapshot = V16PodU128::new(snapshot_residual);
         self.header.payout_snapshot_pnl_pos_tot = V16PodU128::new(self.junior_claim_bound());
         self.header.payout_snapshot_captured = 1;
+        // C38-3: snapshot_residual is intentionally captured once. To keep
+        // payouts solvent, we must hold an invariant that vault never drops
+        // below snapshot_residual minus payouts already disbursed. That
+        // invariant is enforced in validate_shape (see post_snapshot_vault_floor
+        // check) — any residual-debiting path (bankruptcy, cranker reward,
+        // c_tot drift) MUST validate_shape() before returning.
         // ... + new validate_post_snapshot_vault_floor() invariant function
```

VERIFICATION VERDICT

V1 (2026-05-26 08:24:22Z) — SKIP_NO_FLIPPING_POC — patch option (b) adds new validate_post_snapshot_vault_floor invariant; narrative concedes 'no existing test pins' the behavior change. Source-pin POC at line 427 deliberately preserved by option (b), runtime POC at line 360 also preserved. Bug is documented in poc_c38_3_snapshot_residual_is_frozen_once_captured; remediation requires a new invariant test that no existing POC covers.

FINDING 32 / 36

LOW | GRIEFING | F32 | C10-3 | PATCH FLIPS POC

handle_update_base_unit_mints rotates mints with no vault-empty preflight — strands old-mint deposits

AFFECTED CODE

`percolator-prog/src/v16_program.rs:7481-7513`

BUG

Handler rotates cfg.collateral_mint + cfg.secondary_collateral_mint with zero inspection of vault SPL balances. After swap, is_withdrawable_collateral_mint (10411) no longer returns true for the original mint, so any vault balance in the old mint is non-withdrawable.

IMPACT

Admin (base_unit_authority) can freeze active deposits by rotating mints. No theft — funds stay in the vault — but they become unreachable through normal withdraw paths until the old mint is re-added.

STATUS

griefing (admin-only)

EXISTING POC

`percolator-prog/tests/litesvm_l5_misc.rs:418 + source-pin v16_low_source_pins.rs:73`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -7504,9 +7505,18 @@
         verify_mint(primary_mint_ai)?;
         verify_mint(secondary_mint_ai)?;

         let (mut cfg, _, _, _) =
             state::read_market_config_mode_and_capacity(&market_ai.try_borrow_data()?)?;
         expect_live_authority(&cfg.base_unit_authority, authority.key)?;
+        // Refuse to rotate if any atoms remain in the current primary vault.
+        // Without this the old mint becomes non-withdrawable (10411) and the
+        // SPL balance is stranded. See C10-3.
+        let (vault_authority, _) = derive_vault_authority(program_id, market_ai.key);
+        let current_primary = primary_collateral_mint(&cfg);
+        verify_vault_token_account(current_mint_vault, &vault_authority, &current_primary)?;
+        if unpack_token_account(current_mint_vault)?.amount != 0 {
+            return Err(PercolatorError::InvalidVaultAccount.into());
+        }
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC handle_update_base_unit_mints vault preflight → source-pin FAILED at line 86 (handler contains unpack_token_account).

FINDING 33 / 36

LOW | GRIEFING | F33 | C24-5 | PATCH FLIPS POC

ASSET_ACTION_RETIRE writes bare user-supplied now_slot into group.header.current_slot — asymmetric with wrapped reads in same arm

AFFECTED CODE

`percolator-prog/src/v16_program.rs:7946-7964 (raw write at 7964)`

BUG

The same RETIRE arm uses authenticated_slot_or_fallback(now_slot) for the shutdown_asset_empty_and_matured_at_slot_view call (line 7943) but then writes the bare user-supplied now_slot into group.header.current_slot at line 7964. Asymmetric — read-side guarded, write-side raw.

IMPACT

Admin (or asset_authority via RETIRE) can force header.current_slot forward to an arbitrary now_slot value as long as now_slot >= header.current_slot.get() (the line-7946 stale check). The slot scalar is consumed by downstream gates (e.g. slot_last < current_slot loss-stale comparisons) so a privileged-but-skewed write can flip downstream state.

STATUS

griefing (privileged), undocumented

EXISTING POC

`percolator-prog/tests/v16_low_source_pins.rs:118`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -7961,7 +7961,9 @@
                                cfg.free_market_slot_count = cfg.free_market_slot_count
                                    .checked_add(1)
                                    .ok_or(PercolatorError::EngineCounterOverflow)?;
-                               group.header.current_slot = percolator::V16PodU64::new(now_slot);
+                               group.header.current_slot = percolator::V16PodU64::new(
+                                   authenticated_slot_or_fallback(now_slot),
+                               );
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC RETIRE arm authenticated_slot_or_fallback wrap → source-pin FAILED at line 119 (raw write absent).

FINDING 34 / 36

LOW | F34 | C27-2 | PATCH FLIPS POC

ClosePortfolio enables cross-account-type re-init as InsuranceLedger via magic-only check

AFFECTED CODE

`percolator-prog/src/v16_program.rs:359-361 (state::is_initialized) + ClosePortfolio handler + SyncInsuranceLedger init path`

BUG

`state::is_initialized` checks only the magic u64 at offset 0. ClosePortfolio zeroes the buffer (including magic), so is_initialized() returns false afterward. SyncInsuranceLedger's read_or_new_insurance_ledger then takes the "not initialized" branch and re-types the same Solana account as an InsuranceLedger.

IMPACT

A closed portfolio key can be re-purposed as an insurance ledger by anyone who can call SyncInsuranceLedger with that account. The behavioral POC confirms the account passes read_insurance_ledger post-conversion. No theft path identified yet (the new InsuranceLedger carries the caller-provided authority), but the kind-confusion is a foothold for future composability bugs.

STATUS

latent / griefing

EXISTING POC

`percolator-prog/tests/litesvm_l5_misc.rs:485`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -355,9 +355,17 @@
     #[inline]
     pub fn is_initialized(data: &[u8]) -> bool {
-        data.len() ≥ HEADER_LEN && read_u64(data, 0).ok() == Some(MAGIC)
+        // Treat a "closed-sentinel" account as still initialized so subsequent
+        // init paths refuse to re-type it. See C27-2.
+        if data.len() ≥ HEADER_LEN && read_u64(data, 0).ok() == Some(CLOSED_SENTINEL_MAGIC) {
+            return true;
+        }
+        data.len() ≥ HEADER_LEN && read_u64(data, 0).ok() == Some(MAGIC)
     }
+
+    pub const CLOSED_SENTINEL_MAGIC: u64 = 0x434C_4F53_4544_5F31; // "CLOSED_1"

// Companion: ClosePortfolio writes the sentinel as its last step:
//     data.fill(0);
//     data[..8].copy_from_slice(&state::CLOSED_SENTINEL_MAGIC.to_le_bytes());
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC CLOSED_SENTINEL_MAGIC in is_initialized + write sentinel in close → litesvm POC FAILED at line 495 (close_portfolio must zero data buffer assertion broke). .so rebuilt with cargo build-sbf.

FINDING 35 / 36

LOW | F35 | C33-5 | PATCH FLIPS POC

handle_swap_secondary_for_primary moves SPL tokens between vaults but does not update group.header.vault

AFFECTED CODE

`percolator-prog/src/v16_program.rs:7515-7572`

BUG

Handler moves SPL tokens between primary and secondary vaults but never opens a mutable market view. The engine's group.header.vault (which is denominated in primary collateral) sees no debit even though the SPL primary-vault balance increased and the SPL secondary-vault balance dropped.

IMPACT

SPL balances drift above the engine's accounted scalar. Down-stream insurance / withdraw paths bound by header.vault underestimate the real reserve. Reversible (no theft path under current accounts), but accounting invariants are violated.

STATUS

latent (accounting drift, no immediate drain vector)

EXISTING POC

`percolator-prog/tests/v16_low_source_pins.rs:155`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -7549,12 +7549,28 @@
+        // Reflect the primary-collateral inflow in the engine's accounted
+        // vault scalar. Without this, group.header.vault drifts below the
+        // SPL primary-vault balance. See C33-5.
+        {
+            let mut market_data = market_ai.try_borrow_mut_data()?;
+            let (_, mut group) = state::market_view_mut(&mut market_data)?;
+            group.header.vault = percolator::V16PodU128::new(
+                group.header.vault.get()
+                    .checked_add(amount)
+                    .ok_or(PercolatorError::EngineCounterOverflow)?,
+            );
+            group.validate_shape().map_err(map_v16_error)?;
+        }
```

VERIFICATION VERDICT

V6 — FIX_VERIFIED_PATCH_FLIPS_POC handle_swap_secondary_for_primary group.header.vault update → source-pin FAILED at line 164 (handler contains group.header.vault =).

FINDING 36 / 36

LOW | GRIEFING | F36 | C34-C | PATCH FLIPS POC

handle_close_resolved expect_signer conditional on force_close_delay — post-delay any caller can finalize victim's close

AFFECTED CODE

`percolator-prog/src/v16_program.rs:8905-9022 (conditional at :8931-8937)`

BUG

`expect_signer(owner)` is gated inside `if cfg.force_close_delay_slots ≠ 0 && elapsed < force_close_delay_slots`. Once elapsed >= force_close_delay_slots (or if force_close_delay_slots == 0), the signer check is skipped, and any caller can submit close on the victim's behalf. The handler still derives dest_token from accounts[3] and the verified token account ownership check (verify_withdrawable_token_accounts, line 9001) ensures payout lands at the legitimate owner's ATA — so no theft.

IMPACT

Force-claim griefing. Anyone can finalize a victim's portfolio close at the moment force-close eligibility opens, denying the victim the ability to time their own close (e.g., to pair with another tx, to wait for a better swap rate downstream, to keep the position open for tax purposes).

STATUS

griefing-only — funds are not redirected

EXISTING POC

`percolator-prog/tests/v16_low_source_pins.rs:198`

FIX (UNIFIED DIFF)

```
--- a/percolator-prog/src/v16_program.rs
+++ b/percolator-prog/src/v16_program.rs
@@ -8928,12 +8928,11 @@
             if group.header.mode ≠ 1 {
                 return Err(PercolatorError::EngineLockActive.into());
             }
-            if cfg.force_close_delay_slots ≠ 0
-                && authenticated_market_slot_or_fallback_view(&group)
-                    .saturating_sub(group.header.resolved_slot.get())
-                    < cfg.force_close_delay_slots
-            {
-                expect_signer(owner)?;
-            }
+            // C34-C: always require the portfolio owner to authorize their own
+            // close. The prior gating left a force-claim griefing window once
+            // the delay elapsed.
+            expect_signer(owner)?;
```

VERIFICATION VERDICT

V5 (2026-05-26 08:15:39Z) — FLIPPED — patch removes the conditional gate entirely, pin trips at line 224. Compile clean.

---

99 — APPENDIX

A1 — Dedup methodology (39 → 36)

Cluster merges reduced 39 candidates to 36:

MERGED INTO	FROM	RATIONALE
`F1`	C6-1 + C6-2 + C6-3	Three symptoms of one bug — wrapper handle_close_resolved re-implements inline instead of delegating to engine. One patch, three asserted defects.
`F4`	C3-4 + C3-5	Symmetric pair — same write-only counter pattern on insurance side (C3-4) and counterparty side (C3-5). Two patches, one disclosure narrative.
`F13`	C20-1 + C20-2	Two adjacent push handlers (auth_mark + ewma_mark) with identical fix shape — both omit asset.effective_price refresh. One disclosure.
`F16`	C32-1 + C32-3	Two facets of one LATENT problem — trade CU scales linearly with max_market_slots. Wrapper cap blocks today; one latent advisory.

A2 — Test infrastructure notes

All POCs are reproducible from the bounty repo:

- **L2 source-pin tests**: `cargo test --features test --test <pin_file>`
- **L3 Kani**: requires WSL Ubuntu + cargo-kani 0.67.0 + CBMC + cadical. `cargo kani --tests --features fuzz --harness <harness_name>`
- **L4 LiteSVM (V16CuEnv)**: `cargo test --test v16_cu <test_name> -- --nocapture`. BPF build via `cargo build-sbf --no-default-features`.
- **Cargo git cache caveat** — `percolator-prog`'s `Cargo.toml` pins engine as a git dep at rev `23de295`. Local edits to `percolator/src/v16.rs` are NOT picked up by wrapper-side test compilation unless the same edit is applied to `~/.cargo/git/checkouts/percolator-*/23de295/src/v16.rs` AND `percolator-prog/target/debug/deps/libpercolator-*.rlib` is deleted.

A3 — Disclosure ordering guidance

1. **F1 + F2 (CRITICAL)** are headline. F1's positive-PnL-stuck is the most visceral; F2's health-cert bypass is the most catastrophic if exploitable.
2. **F4 (impaired liens)** is the strongest "spec violation" narrative since both sides of a symmetric primitive are missing.

3. **F18 + F19 + F31 (bad-debt family)** could be presented together in a single issue body — same area, three distinct mechanisms.
4. **F14 + F15 (matcher CPI)** can share one issue body since both bound to `handle_trade_cpi`.
5. **F24 + F25 (EWMA poisoning)** form an attack chain; cross-reference in disclosure.

A4 — Authorship + verification chain

STAGE	AUTHOR	VERIFIED BY
L1 hunt	Jelleo hunt agents (38 surfaces beyond anti-scope)	Operator (Kirill)
L1.5 triage	4-way reviewer debate per candidate	Operator
L2 POCs	POC author agents (LiteSVM + source-pin)	Compiler + test runner
L3 Kani	K1-K6 parallel agents (CBMC + cadical)	Kani VERIFIED output
L4 LiteSVM	L4 agents (V16CuEnv harness)	Operator review of test output
L5 narratives	Per-cluster narrative agents (N1-N6)	Operator review
L6 patch verify	V1-V6 verification agents	POC flip PASS→FAIL
Report assembly	This document	Operator cover-to-cover read

END OF REPORT · 36 unique findings · 2026-05-26