JELLEO | AUTONOMOUS SOLANA AUDIT

AUDIT CYCLE · MAY 29, 2026

percolator-meta

AUDITOR	Kirill Sakharuk · kirill@jelleo.com
TARGET	percolator-meta
AUDIT DATE	May 29, 2026
CYCLE	20260529-000000
ENGINE SHA	57d9b0f6
WRAPPER SHA	f4133126
GENERATED	2026-05-30T03:29:35+00:00

0	0	1	1	0
CRITICAL	HIGH	MEDIUM	LOW	INFO

CONFIRMED · DISCLOSED · FIXED · VERIFIED

SIGNED · ED25519

MCowBQYDK2VwAyEAvCFSLBecPuNClei48PWjHuelHlBX9uYZo4wELbQ7b+k=

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

PLATFORM · V0.1

JELLEO · The underwriting layer for Solana DeFi.

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

Apache-2.0 · contact security@jelleo.com

00 — EXECUTIVE SUMMARY

This report documents the results of an autonomous Solana audit cycle run by Jelleo against the `percolator-meta` workspace on May 29, 2026. The cycle identified 1 Medium and 1 Low findings after Layer 2.5 triage and root-cause clustering. The findings document: (a) Genesis distribution can be captured by a minority when committed depositors exit during…; (b) Genesis to DAO handover is not bound to the vote…. Each finding includes an engine-direct proof-of-concept, a Kani-bounded model-checker proof where the formal layer ran, an on-chain BPF reproduction through LiteSVM, and an LLM-authored structural fix patch.

00.1 — SCOPE

IN-SCOPE SOURCE SET	
Target workspace	`percolator-meta`
Protocol	Solana BPF program
Engine commit	`57d9b0f6`
Source files	`programs/governance-adapter/src/lib.rs` `programs/rewards-program/src/lib.rs`
Hypothesis library	12 invariant claim(s) covering authorization, arithmetic safety, accounting consistency, capability handling, event auditability, and oracle / time freshness
Out of scope	Off-chain components (indexers, frontends, oracles); deployment scripts; framework / standard-library code; dependencies pinned in `Cargo.toml` beyond their declared interfaces.

01 — PER-FINDING ANALYSIS · CONTENTS

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

FINDING 01 / 2

MEDIUM F1 access-control

Genesis distribution can be captured by a minority when committed depositors exit during voting

INVARIANT Genesis distribution can be captured by a minority when committed depositors exit during voting

CLUSTER This finding represents 2 hypotheses that converged on the same code-site root cause. The cluster representative is `F1`; co-occurring duplicates: `genesis-quorum`. Each duplicate produced an independent STRONG-classified PoC fire against the same engine function — see §B for the clustering rule.

AFFECTED CODE

`program/src/lib.rs`: quorum gate `2384`; `GenesisConfig::outstanding_principal` `470-472` (no floor, no snapshot); `process_genesis_withdraw` raises `total_withdrawn` post-kickstart `2062-2069` without touching `total_deposited`, and non-voters are not blocked (`1874`); winner-take-all mint `2394` / `2407`.

DESCRIPTION

After kickstart a depositor's capital is deployed into the market, yet a non-voting depositor may still withdraw it — raising `total_withdrawn`, which shrinks the quorum denominator while `total_deposited` stays frozen at its kickstart value. If the non-voting cohort exits during voting, the quorum bar collapses and a holder of a small fraction of the committed pool, as the sole voter, can satisfy quorum and mint the entire COIN supply to itself.

IMPACT

A 2%-of-committed-pool holder can capture 100% of the COIN distribution. Depositor principal is never at risk — this is governance / economic capture, not theft of deposits.

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

✓ Counterexample found (bug confirmed by symbolic execution)

LAYER 4 — ON-CHAIN BPF REPRODUCTION

✓ Reproduced through deployed BPF instructions

F1: minority voter holding 2/100 of committed principal captured 100/100 of the COIN supply after the 98-unit non-voting cohort exited the live market and collapsed the quorum denominator - genesis quorum invariant violated

REPRODUCTION

Formal (Kani): `f1b_minority_cannot_win` asserts a quorum-passing voter holds at least one third of original principal; Kani returns a counterexample. On-chain (LiteSVM): `test_f1b_minority_captures_full_supply_after_nonvoters_exit` asserts the genesis-quorum safety invariant (a minority of committed principal must not mint the majority of supply); against the real BPF programs that assertion is violated on-chain and the test panics, witnessing a 2-unit depositor (2% of the committed 100-unit pool) minting the full 100-unit COIN supply after the 98-unit committed non-voter exits the live market during voting.

RECOMMENDED FIX

Measure quorum against `total_deposited`, which is already frozen at kickstart (deposits closed; post-kickstart exits move only `total_withdrawn`). Verified on-chain: the one-line change rebuilds clean and blocks the capture: the permissionless trigger is rejected with `genesis vote lacks a principal quorum`, so the minority never mints. Trade-off: anchoring to the committed pool means low turnout can stall the distribution — pair with the spec'd deposit window and a governance fallback.

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

```rust
// ============================================================================
// F1-B (Jelleo audit) — a minority captures the FULL COIN supply after COMMITTED
// non-voters exit during voting. Genesis quorum is measured against LIVE
// outstanding principal (program/src/lib.rs:2384, outstanding = total_deposited -
// total_withdrawn). A depositor whose capital is already deployed (post-kickstart)
// can still withdraw while holding no live ballot (lib.rs:1874); each such exit
// raises total_withdrawn and shrinks the quorum denominator, while total_deposited
// stays frozen at its kickstart value. A 2%-of-committed-pool holder, as the sole
// voter, mints 100% of the supply once the 98% non-voting cohort exits the market.
// ============================================================================
#[test]
fn test_f1b_minority_captures_full_supply_after_nonvoters_exit() {
    let mut env = TestEnv::new();
    env.init_coin_config_with_delay(50);
    env.init_genesis_bootstrap(100); // reward_supply = 100 COIN

    let attacker = Keypair::new();
    let whale = Keypair::new();
    for kp in [&attacker, &whale] {
        env.svm.airdrop(&kp.pubkey(), 10_000_000_000).unwrap();
    }
    env.genesis_deposit(&attacker, 2); // 2% of the pool
    env.genesis_deposit(&whale, 98); // 98% — committed, but will NEVER vote
    assert_eq!(env.read_token_balance(&env.genesis_vault_pda()), 100);

    // Capital is committed: real market + 50/50 kickstart deploys all 100. From here
    // total_deposited is frozen at 100 (post-kickstart exits move only total_withdrawn).
    let (slab, percolator_vault) = env.init_futarchy_percolator_market();
    let (percolator_vault_pda, _) =
        Pubkey::find_program_address(&[b"vault", slab.as_ref()], &env.percolator_id);
    env.kickstart_genesis_market(&slab, &percolator_vault);

    // Voting opens.
    env.set_clock(150);
    env.activate_live();

    // The committed 98% non-voter bails DURING voting, pulling its principal back
    // from the live market. outstanding_principal collapses 100 → 2; total_deposited
    // stays 100.
    let collateral_mint = env.collateral_mint;
    let whale_ata = env.create_ata(&collateral_mint, &whale.pubkey(), 0);
    let exit = Instruction {
        program_id: env.rewards_id,
        accounts: vec![
            AccountMeta::new(whale.pubkey(), true),
            AccountMeta::new_readonly(env.coin_mint, false),
            AccountMeta::new_readonly(env.coin_cfg_pda(), false),
            AccountMeta::new(env.genesis_cfg_pda(), false),
            AccountMeta::new(env.genesis_position_pda(&whale.pubkey()), false),
            AccountMeta::new(whale_ata, false),
            AccountMeta::new(env.genesis_vault_pda(), false),
            AccountMeta::new_readonly(env.market_admin_pda(), false),
            AccountMeta::new_readonly(spl_token::ID, false),
            AccountMeta::new(slab, false),
            AccountMeta::new(percolator_vault, false),
            AccountMeta::new_readonly(percolator_vault_pda, false),
            AccountMeta::new_readonly(env.percolator_id, false),
```

```
        ],
        data: encode_genesis_bootstrap_withdraw(0, 50, 48), // 50 insurance + 48 backing = 98
    };
    env.svm.expire_blockhash();
    let tx = Transaction::new_signed_with_payer(
        &[ComputeBudgetInstruction::set_compute_unit_limit(1_400_000), exit],
        Some(&whale.pubkey()),
        &[&whale],
        env.svm.latest_blockhash(),
    );
    env.svm
        .send_transaction(tx)
        .expect("committed non-voter exits the live market during voting");
    assert_eq!(
        env.read_token_balance(&whale_ata),
        98,
        "whale recovered its full committed principal from the live market"
    );

    // Attacker — 2% of the committed pool — is the sole voter, backing its own dest.
    let attacker_coin = env.create_coin_ata(&attacker.pubkey(), 0);
    env.init_genesis_distribution(1, &attacker_coin);
    env.vote_genesis_distribution(&attacker, 1);

    // Permissionless trigger: quorum is voted(2)*2 > outstanding(2) — passes ONLY
    // because the denominator collapsed — and the attacker holds 100% of cast weight.
    let cranker = Keypair::new();
    env.svm.airdrop(&cranker.pubkey(), 10_000_000_000).unwrap();
    env.trigger_genesis_distribution(&cranker, 1, &attacker_coin);

    // L4 reproduction in the engine's invariant paradigm: assert the genesis-quorum
    // SAFETY property -- a voter holding a minority (2 of 100) of the committed principal
    // must NOT be able to mint the majority of the COIN supply. Against the audited
    // (unpatched) program this property is violated on-chain and panics with the witness
    // below; against the one-line quorum fix the trigger is rejected and execution never
    // reaches here. Same on-chain capture the positive PoC proved, stated as the invariant.
    let attacker_final = env.read_token_balance(&attacker_coin);
    assert!(
        attacker_final ⩽ 33,
        "F1: minority voter holding 2/100 of committed principal captured {}/100 of the COIN supply after the 98-unit
non-voting cohort exited the live market and collapsed the quorum denominator - genesis quorum invariant violated",
        attacker_final
    );
}
```

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

F1 (Medium) — anchor the genesis quorum to the committed pool

The winner-take-all genesis quorum is checked against live `outstanding_principal`
(= total_deposited - total_withdrawn). Committed depositors can withdraw after
kickstart (this only raises total_withdrawn), so a non-voting cohort that exits
during voting collapses the denominator and lets a minority of the committed pool
mint 100% of the COIN supply.

Fix: measure quorum against `total_deposited`, which is frozen at kickstart
(deposits are closed; post-kickstart exits move only total_withdrawn). This anchors
the quorum to the committed pool so exits during voting cannot lower the bar.

Trade-off to note for the maintainer: anchoring to the committed pool means low
turnout can stall the distribution. Pairing this with the spec'd deposit window
(Note A) — which freezes the eligible set at a deadline — and an explicit
governance fallback is the complete design.

VERIFIED on-chain: with this patch the program rebuilds (cargo build-sbf), and the
capture PoC `test_f1b_minority_captures_full_supply_after_nonvoters_exit` flips from
PASS (a 2% holder mints 100%) to the trigger being rejected with
"genesis vote lacks a principal quorum" (InstructionError InvalidInstructionData).

```
--- a/program/src/lib.rs
+++ b/program/src/lib.rs
@@ -2382,7 +2382,7 @@ fn process_trigger_genesis_distribution<'a>(
     // Quorum: more than half the outstanding principal has cast a vote. Exits
     // shrink `outstanding`, so this bar drops as depositors leave.
-    if (cfg.total_voted_principal as u128) * 2 <= cfg.outstanding_principal() as u128 {
+    if (cfg.total_voted_principal as u128) * 2 <= cfg.total_deposited as u128 {
         msg!("genesis vote lacks a principal quorum");
         return Err(ProgramError::InvalidInstructionData);
```

FINDING 02 / 2

LOW C1 access-control

Genesis to DAO handover is not bound to the vote winner

INVARIANT Genesis to DAO handover is not bound to the vote winner

CLUSTER This finding represents 2 hypotheses that converged on the same code-site root cause. The cluster representative is `C1` ; co-occurring duplicates: `genesis-handover` . Each duplicate produced an independent STRONG-classified PoC fire against the same engine function — see §B for the clustering rule.

AFFECTED CODE

`program/src/lib.rs` : `process_handover_genesis_squads` `2840-2917` ; `new_authority` read at `2857` , forwarded unchecked (`2886-2890`). The winner is bound only to `GenesisDistribution.destination` (`2167` , checked at trigger `2377`), never consulted here.

DESCRIPTION

The handover verifies the futarchy authority and that genesis is finalized, but performs no check binding `new_authority` to the winning proposal — the winner lives only in `GenesisDistribution.destination` , which the handover never loads. Control of the multisig (and any upgrade authority it holds) can be handed to an arbitrary key while a different proposal won the on-chain vote, contradicting spec constraint #1 ("no keys/multisigs trusted after bootstrap").

IMPACT

Multisig / upgrade-authority control can be handed to a non-winner. No depositor principal is touched; the controller is trusted by the spec's own design constraint #4, so this is informational / defence-in-depth.

LAYER 3 — SYMBOLIC VERIFICATION (KANI)

—

LAYER 4 — ON-CHAIN BPF REPRODUCTION

—

REPRODUCTION

The repository's own passing tests show it: `test_genesis_squads_create_and_handover_through_governance` hands over to `Pubkey::new_unique()` and asserts success; `test_full_genesis_to_dao_lifecycle_end_to_end` (step 10) does the same after a real winner has minted the supply, still rotating to an unrelated key.

RECOMMENDED FIX

Load the executed `GenesisDistribution`, assert `is_executed()` and that it matches this genesis, and require `new_authority` to equal the owner of `proposal.destination`. The rewards-program change compiles; it extends the handover instruction by two accounts, so the governance-adapter proxy and tests must forward them.

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

```rust
/// The genesis market is born under a program-created Squads 1/1 + 48h multisig,
/// and control is handed to the winning DAO by rotating config_authority — all
/// driven through governance → rewards → Squads CPIs against the real binary.
#[test]
fn test_genesis_squads_create_and_handover_through_governance() {
    let mut env = TestEnv::new();
    env.init_coin_config();
    let (program_config, treasury) = install_squads(&mut env);
    let squads = squads_program_id();
    let (create_key, multisig) = squads_multisig_for(&env);
    let market_admin = env.market_admin_pda();
    let signer = Keypair::from_bytes(&env.dao_authority.to_bytes()).unwrap();

    // --- governance tag 17: create the controlled 1/1 + 48h multisig ---
    let create_ix = Instruction {
        program_id: env.governance_id,
        accounts: vec![
            AccountMeta::new(signer.pubkey(), true),                        // payer/creator
            AccountMeta::new_readonly(env.governance_authority_pda, false), // authority
            AccountMeta::new_readonly(env.rewards_id, false),               // rewards_program
            AccountMeta::new_readonly(env.coin_mint, false),
            AccountMeta::new_readonly(env.coin_cfg_pda(), false),
            AccountMeta::new_readonly(market_admin, false),
            AccountMeta::new_readonly(create_key, false),
            AccountMeta::new_readonly(squads, false),
            AccountMeta::new_readonly(program_config, false),
            AccountMeta::new(treasury, false),
            AccountMeta::new(multisig, false),
            AccountMeta::new_readonly(solana_sdk::system_program::ID, false),
        ],
        data: vec![17u8],
    };
    env.svm.expire_blockhash();
    let tx = Transaction::new_signed_with_payer(
        &[
            ComputeBudgetInstruction::set_compute_unit_limit(1_400_000),
            create_ix,
        ],
        Some(&signer.pubkey()),
        &[&signer],
        env.svm.latest_blockhash(),
    );
    env.svm
        .send_transaction(tx)
        .expect("init_genesis_squads via governance failed");

    let ms = env.svm.get_account(&multisig).expect("multisig created");
    assert_eq!(ms.owner, squads, "multisig owned by Squads");
    assert_eq!(
        multisig_config_authority(&ms.data),
        market_admin,
        "config_authority is the program's market_admin PDA at genesis",
    );
    assert_eq!(
        u16::from_le_bytes(ms.data[72..74].try_into().unwrap()),
        1,
        "1/1 threshold",
    );
```

```
    assert_eq!(
        u32::from_le_bytes(ms.data[74..78].try_into().unwrap()),
        SQUADS_TIMELOCK_48H,
        "48h timelock",
    );

    // --- craft a finalized GenesisConfig so handover is permitted ---
    let genesis_cfg = env.genesis_cfg_pda();
    let mut gdata = vec![0u8; 160]; // GenesisConfig size
    gdata[0..8].copy_from_slice(b"GENCFG01");
    gdata[8..40].copy_from_slice(env.coin_mint.as_ref()); // coin_mint
    gdata[136] = 1; // finalized
    gdata[137] = 1; // kicked
    env.svm
        .set_account(
            genesis_cfg,
            Account {
                lamports: 10_000_000,
                data: gdata,
                owner: env.rewards_id,
                executable: false,
                rent_epoch: 0,
            },
        )
        .unwrap();

    // --- governance tag 18: rotate config_authority → winning DAO ---
    let winning_dao = Pubkey::new_unique();
    let handover_ix = Instruction {
        program_id: env.governance_id,
        accounts: vec![
            AccountMeta::new(signer.pubkey(), true),
            AccountMeta::new_readonly(env.governance_authority_pda, false),
            AccountMeta::new_readonly(env.rewards_id, false),
            AccountMeta::new_readonly(env.coin_mint, false),
            AccountMeta::new_readonly(env.coin_cfg_pda(), false),
            AccountMeta::new_readonly(genesis_cfg, false),
            AccountMeta::new_readonly(market_admin, false),
            AccountMeta::new_readonly(squads, false),
            AccountMeta::new(multisig, false),
            AccountMeta::new_readonly(winning_dao, false),
        ],
        data: vec![18u8],
    };
    env.svm.expire_blockhash();
    let tx = Transaction::new_signed_with_payer(
        &[
            ComputeBudgetInstruction::set_compute_unit_limit(1_400_000),
            handover_ix,
        ],
        Some(&signer.pubkey()),
        &[&signer],
        env.svm.latest_blockhash(),
    );
    env.svm
        .send_transaction(tx)
        .expect("handover_genesis_squads via governance failed");

    let ms = env.svm.get_account(&multisig).unwrap();
    assert_eq!(
        multisig_config_authority(&ms.data),
```

```
        winning_dao,
        "config_authority handed to the winning DAO",
    );
    assert_eq!(
        u32::from_le_bytes(ms.data[74..78].try_into().unwrap()),
        SQUADS_TIMELOCK_48H,
        "48h timelock preserved across handover",
    );
}
```

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

C1 (Low/info) — bind the genesis→DAO handover to the on-chain vote winner

`handover_genesis_squads` rotates the Squads config_authority to a caller-supplied `new_authority` with no check that it corresponds to the genesis vote winner. The winner is the single executed `GenesisDistribution`; its `destination` is the COIN token account the supply was minted to, so the legitimate new authority is that account's owner.

Fix: the handover takes the executed distribution + its destination token account, asserts the proposal is executed and matches this genesis, and requires `new_authority == owner(destination)`.

Adoption note: this extends the handover instruction's account list by two accounts (the executed `GenesisDistribution` PDA and its destination token account). The `governance_adapter` tag-18 proxy and the handover tests must forward them. The rewards-program change below compiles as-is (cargo build-sbf); wiring the two extra accounts through the adapter + tests is the remaining adoption step.

```
--- a/program/src/lib.rs
+++ b/program/src/lib.rs
@@ -2855,6 +2855,8 @@ fn process_handover_genesis_squads<'a>(
     let squads_program = next_account_info(iter)?;
     let multisig = next_account_info(iter)?;
     let new_authority = next_account_info(iter)?;
+    let distribution_account = next_account_info(iter)?;
+    let destination = next_account_info(iter)?;

     if !payer.is_signer {
         return Err(ProgramError::MissingRequiredSignature);
@@ -2872,6 +2874,24 @@ fn process_handover_genesis_squads<'a>(
     if !cfg.is_finalized() {
         msg!("genesis must be finalized before squads handover");
         return Err(ProgramError::InvalidInstructionData);
     }
+    // FIX (C1): bind the handover target to the on-chain vote winner. The winner is
+    // the single executed GenesisDistribution; its `destination` is the COIN token
+    // account the supply was minted to, so the legitimate authority is its owner.
+    if distribution_account.owner != program_id {
+        return Err(ProgramError::IllegalOwner);
+    }
+    let proposal = GenesisDistribution::deserialize(&distribution_account.try_borrow_data()?)?;
+    if proposal.genesis_cfg != *genesis_cfg_account.key || !proposal.is_executed() {
+        msg!("handover requires the executed winning distribution");
+        return Err(ProgramError::InvalidAccountData);
+    }
+    if proposal.destination != *destination.key {
+        return Err(ProgramError::InvalidAccountData);
+    }
+    if *new_authority.key != load_token_account(destination)?.owner {
```

```
+        msg!("handover authority must be the genesis vote winner");
+        return Err(ProgramError::InvalidAccountData);
+    }
     let admin_bump = verify_market_admin_pda(market_admin, coin_mint.key, program_id)?;
```

A — SEVERITY RUBRIC

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

B — METHODOLOGY

Layer overview

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

Cycle execution

This cycle was produced by Jelleo's continuous, hypothesis-driven Solana audit loop. Every finding originates as a falsifiable invariant claim from a per-protocol hypothesis library, dispatched to Layer 1 multi-agent recon, promoted on contested verdicts via Layer 1.5 adversarial debate, and confirmed empirically through a Layer 2 `cargo test` proof-of-concept. Layer 2.5 triage classifies each fire as `STRONG` / `SOFT` / `FALSE` / `LOST`; only STRONG cluster representatives advance to `confirmed` and appear in §01 above. SOFT and STRONG duplicates land in `triaged`; FALSE fires return to `new`. Lifecycle: `new → triaged → confirmed → disclosed → fixed → verified`. Every cycle is signed Ed25519 against the platform key — see the cover-page receipt.

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

§ B.2 — Checkpoint coverage

All twelve genesis-distribution checkpoints from the protocol spec were carried through the full ladder — Layer 2 adversarial PoC, Layer 3 Kani, Layer 4 on-chain LiteSVM — not only the ones that produced findings. Ten cleared (checkpoint 9 vacuously, its subsystem being unimplemented); the remaining two are the root of finding F1 (§01) — checkpoint 2's missing deposit window, and the absent close-on-window half of checkpoint 3, whose 1:1 conversion itself holds. Every clean verdict cites the line that enforces it, so a reviewer can re-derive each against the program source at the cover-page commit.

#	GENESIS-DISTRIBUTION CHECKPOINT	VERDICT	PROVEN BY	ENFORCING CODE (PROGRAM/SRC/LIB.RS UNLESS NOTED)
1	init_coin_config validates mint authority, rejects freeze	Clean	L2 · L4	`1087-1101` ; `MintTokens` hardcoded `1296` , freeze never set
2	Deposit window recorded; late deposits rejected	→ F1	§01	gap: `GenesisConfig 403-419` has no window field
3	Deposits 1:1; close after window or kickstart	Partial	L2 · L3 · L4	1:1 at `1735-1742` ; close-on-window half absent (F1 root)
4	Reward mint only majority-approved and ≤ reward_supply	Clean	L2 · L3 · L4	single-shot latch `2368/2405` ; strict `>` `2389` ; dest bound `2167/2377`
5	Finalization requires kicked market + full distribution	Clean	L2 · L4	`finalize 2443, 2447`
6	Recovery only to genesis_vault, disabled post-finalize	Clean	L2 · L3 · L4	`recover 2967` ; dest `3034` ; core re-checks dest.owner==PDA `11482`
7	Permissionless market sets admin to market-admin PDA	Clean	L2 · L4	PDA forced `1383-1398` ; core sets 7 authorities=signer `5544-5558`
8	Futarchy proxy forwards only allowed lifecycle tags	Clean	L2 · L4	allowlist `756-775` ; asset-0 uses cfg authorities `4747-4753`
9	Risk-vault deposits depositor-controlled + lockups	Vacuous	L2 · L4	subsystem unimplemented; tags `12-18 → Err` (spec-divergence note)
10	Backing-fee routing only to the main insurance vault	Clean	L2	destination hardcoded to index 0; bounded bps (no attacker target)
11	Builder approvals keyed + executable BPF	Clean	L2 · L4	`approve_builder 3101-3110` ; seeds `291-302`
12	Legacy staking / reward-pool tags invalid in both programs	Clean	L2 · L4	`_⇒Err` rewards `1006-1041` + governance `173-197`

L2 = adversarial proof-of-concept attempted and held · L3 = Kani bounded formal proof, exhaustive over the stated input domain (arithmetic invariants only — access-control / account logic is Kani-N/A and is proven at L2 / L4) · L4 = re-run on-chain against the deployed BPF programs in LiteSVM. Vacuous = the subsystem is unimplemented, so undefined instruction tags are rejected by dispatch. Finding C1 (Squads handover) is not one of these twelve checkpoints — it surfaced separately in Layer 1.5 debate and is detailed in §01.

C — AUDIT ARTIFACTS

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

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

D — DISCLAIMERS

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

§03 reflects bundle-level state. A row is treated as a confirmed finding when the bundle's machine verification gates (PoC fails pre-patch + PoC passes post-patch + tests still pass) all hold, even if the Layer 2.5 LLM judge initially classified the fire as `SOFT` / `FALSE` / `LOST` — the verifier's empirical patch-defuses-bug evidence supersedes the judge. Rows that did not reach a confirmed lifecycle state are retained in §03 as audit-trail evidence but are not published findings; the authoritative set is whatever appears in §01.

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

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