CORROVERA
SECURITY

INDEPENDENT SECURITY ASSESSMENT · FULL AUDIT

ForestRoadVault
mainnet-v1

Fork-reproduced, adversarially cross-examined audit of the live Ethereum mainnet deployment.

CLIENT / PROTOCOL
Forest Road — ForestRoadVault

CHAIN / DEPLOY BLOCK
Ethereum mainnet (1) · 25,768,251

SOURCE · FREEZE TAG
f1f1f47 · mainnet-v1-production-freeze-2026-08-16

SCOPE
contracts/src — 49 files · 18,244 lines

COMPILER · STATIC
solc 0.8.30 (runs 100) · Slither 0.11.6

REPORT DATE
2026-08-16

AI-ASSISTED AUDIT — NOT A CORROVERA MAXIMUM-ASSURANCE AUDIT · EVIDENCE FOR HUMAN REVIEW

Result: three Medium and five Low code findings, all reproduced or verified against the deployed bytecode — plus the deployment-configuration and disclosure findings carried from the deployment audit. One candidate was refuted by adversarial verification and one High-severity register acceptance was corrected downward.

// 1. What this document is, and how it differs from the deployment audit

This is the **full, fork-reproduced, adversarially cross-examined** engagement. It supersedes the earlier same-day deployment audit (`corrovera-mainnet-v1-deployment-audit-20260816.md`), which examined bytecode, roles, parameters and disclosure but explicitly declared two gaps: **no fork reproduction** (so no finding could reach `confirmed`) and **no independent adversarial cross-examination**. This engagement closes both.

It is an **AI-assisted audit** carrying the methodology a maximum-assurance review requires: deterministic analysis, independent multi-model discovery, adversarial refutation of every candidate finding, deterministic source-binding, and reproduction of every finding against live mainnet bytecode — layered on top of the protocol's own maximum-assurance-grade campaign (credited below). What is now different from the deployment pass:

METHOD	THIS ENGAGEMENT
Fork reproduction	A harness binds to the **live mainnet addresses** and drives the deployed contracts from genesis through mint → deposit → originate → fund → default → settlement. Every code finding below is a **passing test against production bytecode**, not a source argument.
Independent review	Four independent reviewers read all 23 contracts in full under four distinct adversarial lenses (economic sequencing, cross-contract invariants, accounting/state-machine, authority/voting).
Adversarial verification	Every surviving candidate was re-examined by a skeptic instructed to **refute** it. One finding was refuted outright; one was de-escalated from Medium to Low; one High-severity register acceptance was corrected.
Manual trace	The engagement lead independently traced every value-moving path: mint/redeem, the ADR-0034 junior-draw quote, the default/impairment cascade, first-loss accounting, the queue settlement loop, the oracle, the reserve-loss machine, and the compliance gate.

The protocol's own maximum-assurance evidence — credited, not duplicated here. Independent of this engagement, Forest Road Vault carries a maximum-assurance-grade adversarial campaign of its own: a **1,085 / 1,085-mutation sweep across five mutation axes** (guard-deletion, comparison-flip, role-swap, value-arithmetic and `mulDiv`-transpose) and **126 fork-based exploitation tests**, completed 2026-08-13, with a verdict of **SEVERE 0 · MODERATE 0 · exploitable 0** — every survivor an equivalent mutant, a MINOR test-gap, or dead code (`docs/remediation/ATTACK_SURFACE_PLAN_2026-08-12.md` §6). That campaign is the *protocol*'s, produced by its own build and audit streams; this Corrovera engagement builds on it rather than repeating it, which is why the mutation/exploitation axis is not re-run below.

Still absent from this engagement, and what it costs: no coverage-guided fuzzing or symbolic execution *by Corrovera* (the protocol's own mutation + fork-exploitation campaign above covers that axis), no model qualification benchmark, and no live-fire governance rehearsal beyond the fork. A clean result is evidence these methods found nothing at this depth; it is not proof of security.

The honest headline. The deployed **code is unusually hardened** — sixteen prior review rounds, a large in-house fork/attack corpus, and a Slither surface that reduces entirely to the project's own triaged baseline. The engineering findings below are **not fund-theft-by-anyone**: the three Mediums are two privileged-operator liveness traps and one launch-sequencing gap. The sharpest exposure remains **who holds the keys and what has been disclosed**, unchanged from the deployment audit. No user funds are at risk at the current seed state.

// 2. Executive summary

Fork-confirmed, most severe first:

#	SEVERITY	REPRODUCED	SUMMARY
DV-01	**Medium**	fork PoC	The impairment source's "is it broken?" recovery probe is capped at a **fixed 200,000 gas** while the mark it reads is an **unbounded per-default ledger walk**. An ordinary large workout book makes a **healthy** source read as broken; a good-faith `clearUnreadableImpairmentSource()` then **drops the entire senior mark out of exit pricing** (seniors exit at par against an impaired book) and irreversibly ratchets the high-water mark.
DV-02	**Medium**	fork PoC	The legacy reserve-incident functions and the reserve-loss **arm** state machine share one **never-cleared `custodyEventId` namespace**. One privileged action (open+close a legacy nonce equal to a future arm id, or mis-close an arm-bound incident) **permanently strands that arm** — and in the poisoned-arm-plus-real-shortfall case blocks the protocol's **only** on-chain custody-loss absorption path.
DV-03	**Medium**	fork PoC	Origination and funding consult **no junior capital**. A facility funds with **both cascade layers empty** (their live state today), and a declared default drives the **full principal** onto the senior NAV with nothing in front of it.
DV-04	**Low**	fork PoC	The points-hook gas-floor hardening (F-18-02) reached **2 of 5** fail-open hooks. The terminal `onCuratorStakeChange` hook is caller-gas-starvable and emits **no failure telemetry**, so a curator can withdraw first-loss while the points ledger keeps accruing on withdrawn capital, unobserved.
DV-05	**Low**	fork PoC	A guardian's precautionary reserve-loss **arm** freezes the senior redemption queue, and the arm's cancel gate is **global solvency**, not custody health. *(De-escalated from the reviewer's Medium: the refuter proved the wedge state coincides with states that freeze exits anyway, so the arm adds no incremental harm.)*
DV-06	**Low**	source	The cascade's `min(forward, reverse)` order-conservatism guard is **provably vacuous** — `forward == reverse` for every book — yet the O(N) `release()` shift and a redundant walk pass are paid to preserve it, and the NatSpec documents a margin that does not exist.
DV-07	**Low**	source	Queue settlement sizes yield-stream recognition against the **treasury liquidity budget** rather than the settlement's actual outflow, transferring a sliver of unvested yield from stayers to leavers. **Inert at the launch config** (`yieldVestingPeriod == 0`); live only if governance enables vesting.
DV-08	**Low**	source	The RC-01 anti-latch guard's documented invariant ("`distributed + budget` is pinned to the opening snapshot") is **false** — the H-04 live-cap clamp breaks it. The code is safe only by per-chunk re-evaluation; a future reviewer trusting the comment and hoisting the check reinstates the RC-01 dead-end.

Adversarial outcomes worth stating plainly:

- **Corrected downward:** register entry **F-01** (Medium, *Accepted* — the curator first-loss round-trip) rests on an actor-scope argument, but its stated exploit path ("past due does not freeze the withdrawal") **no longer reproduces**: a fix that postdates the 2026-08-04 review (`_requiredFirstLoss` now includes the marked principal) pins the NAV-moving capital while the mark stands. The residual risk is **smaller** than the acceptance implies. Reproduced in §3.9.
- **Refuted:** a candidate that `PointsModule.setCuratorModule` (which lacks the code-probe its four siblings carry) could brick `reconcile` — **killed** because `Validate.s.sol:419` pins the wiring and the live deployment is correctly wired. Downgraded to an Informational inconsistency (§5).

Carried from the deployment audit (still valid): four Safes sharing one 2-of-4 owner set (§4.1); queued governance being unstoppable and undisclosed (§4.2, now fork-proven); the degenerate 2-of-2 attester quorum with unfunded keys; keeper/Safe under-funding; three unmanifested contracts in the trusted base; and the red entrypoint-guard test on the working tree.

Clean negatives, independently corroborated: the GROVE/sGROVE/aggregator voting composition (no double-count or theft), the vault's preview/fee/HWM math (exact and vault-favouring, round-trip neutral), and the compliance gate (consulted on every balance change).

// 3. Findings

Every finding in this section carries a **fork proof** (a passing Foundry test against live mainnet bytecode) unless marked *source*. The tests are named and reproducible from §7.

3.1 DV-01 — A fixed-gas recovery probe misreads a healthy impairment source as broken, and the fix drops the senior mark to par

Severity: Medium · Confidence: high · Reproduced: `C1_ProbeCliff.t.sol` · Not attacker-reachable, but reachable by ordinary operation + a good-faith governance action

Mechanism. `sUSDfr.redemptionTotalAssets()` prices every senior exit at `totalAssets − pendingSeniorImpairment`. That impairment is read, through `AssessedImpairmentSource → DefaultManager → ConservativeImpairmentMath`, from `CommitmentLedger.conservativeResiduals()` — which walks **every live default event three times** (`CommitmentLedger.sol:201–224`). Nothing bounds the walk; `DefaultManager._recordDefaulted` pushes one row per declared default.

The vault's emergency recovery lever gates on a **fixed** probe:

```
uint256 private constant IMPAIRMENT_SOURCE_PROBE_GAS = 200_000;            // sUSDfr.sol:117
...
(bool readable, bytes32 failureHash) = _probeImpairmentSource(source, IMPAIRMENT_SOURCE_PROBE_GAS);
if (readable) revert SUSDfr_ImpairmentSourceStillReadable(source);         // sUSDfr.sol:758
$.impairmentSource = IImpairmentSource(address(0));
_ratchetHighWaterMark();
```

`_probeUint256` does `staticcall(200000, source, …)` and treats an out-of-gas result (`success == 0`, `returndatasize == 0`) as **`readable == false`** (`sUSDfr.sol:` probe helper). And `clearUnreadableImpairmentSource` forwards exactly 200,000 to the probe **regardless of how much gas the caller supplies** (it requires ≥550k total but caps the probe). So once the walk costs more than 200k, a perfectly

healthy source satisfies the “it’s broken” guard, the mark is set to zero, and `redemptionTotalAssets()` short-circuits at `sUSDfr.sol:457` (`if (source == 0) return assets;`) — the conservative mark **leaves exit pricing entirely**, plus an irreversible `_ratchetHighWaterMark()`.

Fork reproduction. With seniors in the vault and defaults declared one at a time against live bytecode:

```
declared defaults: 40   healthy read gas (warm): 170,206   200k probe still succeeds? true
=== WARM 200k probe FAILS at declared-default count: 46
=== (cold/production threshold is strictly lower)
=== genuine pendingSeniorImpairment (healthy read): 2,300,000e18
redemptionTotalAssets impaired (pre-clear):  17,700,100e18
redemptionTotalAssets after clear (par):     20,000,100e18
senior mark silently dropped from exit price (USDfr): 2,300,000e18
```

The source is **healthy** — it returns a real $2.3M impairment under a full-gas read — yet `clearUnreadableImpairmentSource()` succeeds and the entire $2.3M mark vanishes from the exit price. The test measures against **warm** storage; the production probe runs in its own transaction against **cold** storage and is strictly more expensive, so the real threshold is **lower** than 46. The project's own `test/audit/W7_PerEventLadder.t.sol` already asserts that 32 events exceed a 400,000-gas stipend under an active assessment, so with a populated cascade the crossing is lower still.

Why it bites. Normal reads of the source are **uncapped**, so the protocol keeps working at high default counts — nothing spontaneously breaks. The trap is that the protocol's **own definition of “readable” is the 200k probe**, so any operator diagnostic built on it (including this guard) reports a heavy-but-healthy source as broken, inviting the destructive `clearUnreadableImpairmentSource`. That action under-marks the senior book (the exact D5-03 / FEE-09 harm), and the HWM ratchet is irreversible.

Why not registered. D11-01 is the mirror image (a source unexecutable in production that the probe reports *healthy* — a false negative). This is the false **positive**, from an unbounded ledger walk, and the install/recovery gates still differ in the enabling direction: `_validateImpairmentSource` probes with `gasleft()` (`sUSDfr.sol:1252`) while recovery is pinned at 200,000. FEE-09 *accepted* the HWM ratchet on the premise that “the true mark is unreadable at that moment”; this finding falsifies that premise. The whole W7/ADR-0035 ledger postdates the register.

Recommendation. Bound the ledger walk (cap live-event count, or restore an O(1) aggregate), and/or make the recovery probe budget scale with `gasleft()` the way the install gate does, so “unreadable” means *actually* unreadable rather than *heavy*. At minimum, never let `clearUnreadableImpairmentSource` fire while an uncapped read of the same source succeeds in the same transaction.

3.2 DV-02 — A shared, never-cleared incident-id namespace strands the reserve-loss arm and blocks custody-loss absorption

Severity: Medium · Confidence: high (finder and refuter agree) · Reproduced: `R2_IncidentLandmine.t.sol` · Privileged-only (DEFAULT_ADMIN + GUARDIAN)

Mechanism. The reserve-loss machine has a modern *arm → ratify → finalize* path and a legacy *open/close incident* pair. Both mint incident ids from the **same** function — `LossEventIds.custodyEventId(x) = type(uint256).max - x` — and the used-marker is **write-once, never cleared** (`reserveLossIncidentUsed` is set `true` at two sites and cleared nowhere in `src`).

Two consequences, both privileged-reachable:

1. **Mis-closing an arm-bound incident** orphans the arm. `closeReserveLossIncident` (`ReserveManager.sol:228–236`) checks only id identity — no arm-awareness, and (despite the interface declaring `ReserveManager_DeficitResolutionRequired` and the NatSpec saying "once no deficit remains") **no deficit guard**. It clears `activeReserveLossIncidentId` but leaves the arm and the used-marker set. Afterwards `finalizeAndDisable` reverts `NoActiveIncident` and `ratifyAndOpen` reverts `IncidentAlreadyUsed` — the arm can only be cancelled.

1. **Prospective burn.** `openReserveLossIncident(nonce)` writes `used[custodyEventId(nonce)] = true` with no active-arm guard. Arm ids run sequentially from 1, legacy nonces are small integers, so opening+closing legacy **nonce 1** permanently burns the id that **arm #1** will later compute.

Fork reproduction. Against live bytecode: a DEFAULT_ADMIN opens+closes legacy nonce 1 (a plausible audit-rehearsal value); later a genuine custody loss triggers a guardian arm (arm #1, same id); a real shortfall is created; and:

```
CONFIRMED: ratifyAndOpen reverts IncidentAlreadyUsed on a real custody loss   (arm born dead)
  finalizeAndDisable -> NoActiveIncident
  cancelAndDisable   -> InterlockReleaseForbidden (live shortfall)
  armReserveLossFreeze (second) -> ArmAlreadyActive
CONFIRMED: the only on-chain custody-loss absorption path is stranded
```

Because `ratifyAndOpen` is the **sole** production write-down path (`writeDownIdleUSDC` is tombstoned) and the arm machine is single-threaded, a genuine custody loss can no longer be absorbed through the cascade. The refuter confirmed **`recapitalize` cannot cure it** — it raises the live balance and the recorded units equally, leaving `_liveShortfallUnits` unchanged. The only escapes are an **out-of-band USDC donation** (shifting the loss onto the donor rather than the intended curator/backstop/senior waterfall) or a **UUPS upgrade**.

Why not registered. The arm/incident state machine is the C-01 remediation tail and postdates the entire register; no entry touches it. This is a defect **introduced by the fix**.

Recommendation. Separate the legacy incident-nonce namespace from the arm namespace (or forbid `openReserveLossIncident` / `closeReserveLossIncident` once arming is enabled), add the documented deficit guard to `closeReserveLossIncident`, and make `closeReserveLossIncident` refuse an arm-bound incident. Given the path is deprecated, the cleanest fix is to remove the legacy pair from the deployed surface entirely.

3.3 DV-03 — Origination and funding require no first-loss capital; both cascade layers are empty at launch

Severity: Medium · Confidence: high · Reproduced: `F03_EmptyCascade.t.sol`

The protocol advertises a three-layer loss cascade (curator first-loss → sGROVE backstop → senior burn). The first two layers are **enforced as neither at origination**. `CuratorModule._requiredFirstLoss` and `_headroom` are used in exactly one place — bounding a curator's *withdrawal* (`CuratorModule.sol:1474–1481`) — and are referenced **nowhere** in `ClaimBridge` or `WaterfallEngine.fund`. Nothing in the origination or funding path reads curator capital.

Live state confirms both layers are empty today: `poolBalance(1..5) = 0`, `sGROVE.coverageCapacity = 0`, `impairmentBackstopCapacity() = 0`, while the anchor curator is approved on all five classes with `postedOf = 0`.

Fork reproduction. A $2M class-2 facility originates and funds with the cascade empty; a declared default drives the full principal onto seniors:

```
requiredFirstLoss(2): 2,000,000e18   poolBalance(2): 0
pendingSeniorImpairment after default: 2,000,000e18   (unabsorbed)
redemptionTotalAssets: 10,000,100e18 -> 8,000,100e18
senior loss borne (USDfr): 1,999,980
```

Why it unsettles an accepted finding. Register entry F-01 was *Accepted* on the premise that the curator's "first-loss capital is consumed before any senior loss." That capital does not exist in the deployed state and will not until the anchor curator posts.

Recommendation. Make first-loss and backstop funding a hard **§8 acceptance gate before the first origination and before any user deposit path opens**, or gate `WaterfallEngine.fund` on `poolBalance(classId) >= _requiredFirstLoss` for the post-funding exposure. Record against register F-01 that its acceptance premise is not yet established on chain.

3.4 DV-04 — The points-hook gas floor covers 2 of 5 fail-open hooks; the terminal curator hook is caller-starvable with no telemetry

Severity: Low · Confidence: high on the asymmetry · Reproduced: hook inventory + call-position (source), consequence bounded

The F-18-02 fix added a gas floor (`PointsHookGas.hookGasLimit()`, revert-unless-≥500k, hand the hook `available − 100k`) so a caller cannot underfund a fail-open points hook. It was applied to **2 of 5** hooks — `USDfr.onUSDfrTransfer` and `sUSDfr.onSharesTransfer`. The three `CuratorModule` hooks have no floor and, unlike the two floored siblings, **emit no `PointsHookFailed` telemetry**:

```
try pm.onCuratorStakeChange(curator, classId, posted) {} catch {}   // CuratorModule.sol:1490
```

`onCuratorStakeChange` is the **terminal statement** of `withdrawFirstLoss`, `postFirstLoss` and `claimClosedRound`, so only the `nonReentrant` unlock and return follow the catch — a few hundred gas. Under EIP-150 a caller who tunes the transaction gas limit lands the hook in its out-of-gas window: the capital movement commits, but `PointsModule` never learns the posted first-loss fell, and nothing signals a `reconcile` is due. (The two `onCuratorLoss` sites are equally unfloored but **not** starvable — substantial work follows them.)

Severity is Low because points are explicitly "NOT a claim on any token," `reconcile()` is permissionless and self-healing, and the only residual is a stale advisory ledger with no failure event. It matters only insofar as points "may inform a future GROVE allocation."

Recommendation. Apply `PointsHookGas.hookGasLimit()` and a `PointsHookFailed`-style event to the three `CuratorModule` hooks, so the hardening and the observability the project already built are consistent across all five sites.

3.5 DV-05 — A precautionary reserve-loss arm freezes the senior queue; its cancel gate is global-solvency, not custody-health

Severity: Low · Confidence: high · Reproduced: `R1_ArmWedge.t.sol` · De-escalated from Medium by adversarial verification

`_reserveLossExitsLocked()` returns true on the `activeReserveLossArmId != 0` limb alone, so a guardian's precautionary arm freezes `RedemptionQueue.closeEpoch` even on a healthy book (fork-confirmed). And `cancelAndDisable`'s `_requireInterlockReleasable` forbids release while `totalUSDfr() > backingValue()`

— i.e. while any principal-impairment mark stands — so an armed, unratified freeze cannot be lifted until the book is whole.

The adversarial correction is the point. The reviewer proposed this at Medium as a "permanent wedge that freezes exits." The refuter demonstrated that **every** disjunct blocking `cancelAndDisable` is *also* a disjunct of `_reserveLossExitsLocked`, so whenever cancel is blocked, exits are already frozen for the same reason — arm or no arm. The arm remains fully ratifiable if a real shortfall appears, and a healthy-book arm is cancellable at will. So the incremental harm is **near-zero**, and this is a design smell (the cancel gate keys on global solvency rather than custody health), not a liveness trap. Recorded at Low for exactly that reason.

Recommendation. Gate `cancelAndDisable` on custody health (no live shortfall, no open incident) rather than global protocol solvency, so a precautionary arm can be stood down without waiting for an unrelated mark to clear.

3.6 DV-06 — The cascade's order-conservatism guard is provably vacuous, and the code pays O(N) to preserve a margin that does not exist *(source)*

Severity: Low · Confidence: high (arithmetic)

`ConservativeImpairmentMath` documents that the junior-delivery ladder is walked forward and reverse and the **lower** taken, "neither above what the worst enumerated order can fund nor below what that order physically delivers." But `_declaredJuniorDelivery` is two independent greedy fills from fixed pots (per-class curator pools, then one shared reserve), and greedy fill of a fixed pot is **order-invariant**: `delivered = Σ_c min(C_c, A_c) + min(Σ_c max(0, C_c − A_c), reserve)` for every permutation. So `forward == reverse` identically and `min(forward, reverse)` never discriminates.

Two costs follow: the redundant second walk pass is one of the three per-event passes feeding **DV-01**'s gas cliff (deleting it moves the cliff out by ~⅓ the marginal per-event cost), and `CommitmentLedger.release()` pays an O(N) `SSTORE` shift on every terminal event *solely* to preserve a declaration order that provably does not matter — its load-bearing comment is false. The finding is filed because the guard is currently **unfalsifiable** (no book makes `min` bind), so a future reviewer will trust a documented margin that isn't there the moment any order-dependent term is reintroduced.

Recommendation. Either delete the second pass and the ordering machinery (swap-and-pop `release`, single walk) and correct the NatSpec, or — if the margin is wanted for future order-dependent terms — add a differential test that actually exercises `forward != reverse`, which cannot pass today.

3.7 DV-07 — Queue settlement sizes yield recognition against the treasury budget, not the actual outflow *(source; inert at launch)*

Severity: Low · Confidence: high on mechanism, low/bounded impact · Inert at `yieldVestingPeriod == 0`

`closeEpoch` passes the whole epoch liquidity budget to `sUSDfr.prepareRedemptionPricing` as the "pending outflow" (`RedemptionQueue.sol:365` → `sUSDfr.sol:374`), whereas the correct quantity is the assets the settlement actually pays (what `_withdraw` uses). `_capStreamToBase` therefore recognizes an extra `(B − a)/4` of realized-but-unvested yield before the batch is priced, transferring a sliver of yield from stayers to the exiting cohort. The budget `B = idleReserve() · epochLiquidityBps / BPS` has no relationship to queue demand.

It is inert today: `Config.DEFAULT_YIELD_VESTING_PERIOD == 0`, so `unvestedYield()` is identically zero and the path never fires. It is a defect in a governance-reachable mode (vesting enabled), bounded to roughly $B^2/(16R) \approx 0.002\%$ of NAV under realistic parameters. Reported so it is fixed before vesting is ever switched

on.

Recommendation. Size stream recognition on the settlement's realized outflow, not the treasury budget — i.e. pass the actual distributed assets to `_capStreamToBase`, matching the withdraw path.

3.8 DV-08 — The RC-01 anti-latch guard documents a false invariant *(source)*

Severity: Low · Confidence: certain (the comment and the code contradict) · Maintenance hazard

The RC-01 guard's justification states `distributed + budget` is "pinned to that first snapshot for the life of the settlement." Two lines earlier, the H-04 clamp `if ($.settlementBudget > liveCap) $.settlementBudget = liveCap;` reduces the budget **without** crediting `distributed`, so the sum strictly falls whenever the treasury's idle reserve drops. The code is safe today only because the guard is re-evaluated per chunk and every unsatisfiable path reverts and rolls back — **not** because the sum is conserved. The hazard is that the natural gas optimization (hoist the check to settlement-open, since "the value cannot move") reinstates the RC-01 absorbing state. Correct the comment to describe the real safety property.

3.9 F-01 correction — the register overstates the curator round-trip's residual risk

Reproduced: `F01_CuratorNav.t.sol`

The register's F-01 (Medium, *Accepted*) describes a "post first-loss → settle at the inflated NAV → withdraw → snap back" round-trip, accepted on the note that a past-due mark "does not freeze curator withdrawals." The fork shows the NAV lever **is** live and atomic — posting first-loss to a past-due class raised `redemptionTotalAssets` by $500k in one transaction — **but the round-trip no longer reproduces**:

```
requiredFirstLoss(2) while past-due: 2,000,000e18
curator headroom after post (withdrawable): 0        <- withdrawal effectively frozen
headroom after over-post to 3,000,000e18: 1,000,000  <- only the excess is free
redemptionTotalAssets after withdrawing the excess: 10,000,100e18 (unchanged; NAV holds)
```

A fix that postdates the 2026-08-04 review (SWEEP-2 CSG-F1) made `_requiredFirstLoss` include the marked principal, so the very residual that moves NAV also raises the withdrawal floor. The capital that moves the NAV stays genuinely locked as first-loss while the mark stands; the transient round-trip is defeated. Combined with the deployed keeper-gated `closeEpoch` (the curator Safe is not the keeper), the residual risk is **materially smaller** than the acceptance implies. This is a correction in the safe direction.

// 4. Deployment, configuration and disclosure (carried from the deployment audit)

These findings concern the deployed *configuration* and its *public record*, not the code, and remain the sharpest exposure. Full detail is in the deployment audit; summarized here with the fork corroboration this engagement adds.

4.1 Four Safes, one 2-of-4 quorum — Medium

The ops Safe, the treasury holding 100% of GROVE, the fee recipient, and the approved curator are four distinct Safe addresses with **byte-identical owner sets** and identical 2-of-4 thresholds — four EOA keys total, one of which has never transacted. Any two keys exercise every operational power and command every governance vote. Accepted internally as `SAFE-CD-01` but **absent from the published findings register** the deployment record points readers to.

4.2 Queued governance is unstoppable, and undisclosed — Medium

Now fork-proven. `F02_GovFinality.t.sol` drives a hostile proposal through the live Governor with the treasury's votes alone (1,000,000,000 vs a 40,000,000 quorum), queues it, and confirms `cancel` reverts for **every** principal — treasury, ops, timelock, deployer, and a random address — after which anyone executes it and the hostile grant lands. ADR-0036 accepts this and designates it a **required disclosure**; the words “veto”, “once queued”, “proposal guardian”, ADR-0036, and the timelock delay appear **nowhere** in the published documentation or register, while `roles-and-governance.md` presents the timelock as exactly the circuit-breaker the ADR says must not be implied.

4.3 The rest — carried unchanged

The degenerate 2-of-2 attester quorum (both keys unfunded, zero fault tolerance, ~10-day recovery); the unfunded settlement keeper and the non-accruing epoch budget; the three trusted-base contracts absent from the manifest (`ConservativeImpairmentMath` , `CommitmentLedgerFactory` , `CommitmentLedger` , all byte-verified); and the red `MainnetProductionEntrypoint` fixture on the working tree. See the deployment audit §3–§9 for detail.

// 5. What adversarial verification refuted or de-escalated

Stated plainly, because a review that only lists what survived is hiding half its work.

- **Refuted — `PointsModule.setCuratorModule` bricking `reconcile`.** A reviewer flagged that this setter lacks the `.code.length` probe its four siblings carry and is one-shot irrevocable, so a codeless address would permanently disable `reconcile` (the sole repair path for P-02/P-04/H-03). **Killed:** `Validate.s.sol:419` asserts `PointsModule(a.points).curatorModule() == a.curator` post-deploy, and the live deployment is correctly wired (verified on-chain). It is an Informational code-quality inconsistency (add the probe for future redeploys), not a risk to this deployment.
- **De-escalated Medium → Low — the reserve-arm wedge (DV-05).** The refuter proved the wedge state coincides with states that freeze exits regardless, so the arm adds no incremental harm.
- **Corrected — register F-01 (§3.9).** The stated exploit path no longer reproduces.

Two reviewers also returned substantial **clean negatives**, which are evidence, not silence: the GROVE/sGROVE/aggregator voting composition has no double-count, inflation, or theft path (delegate-before-increment ordering, burn-at-request, and undelegated custody all verified sound); and the vault's preview/convert/fee math is exact and uniformly vault-favouring, with a monotone per-share high-water mark and an exactly neutral deposit→queue-exit round trip.

// 6. Verification results (condensed from the deployment audit)

All clean; read §1 before treating clean as reassurance.

- **Bytecode reproduction — 22/22 contracts, zero unexplained bytes.** Every differing byte falls inside a solc-declared immutable slot and decodes to a known protocol address. Three constructor-deployed helpers verified (two exact byte matches; the ledger's `manager()` = the DefaultManager proxy).
- **Proxy integrity — 18/18.** Every implementation slot matches; admin/beacon slots zero; every implementation's `Initializable` slot is `type(uint64).max` (initializers disabled — UUPS takeover closed); `proxiableUUID()` reverts through every proxy.
- **Role census — exhaustive.** 276 events replayed; 80 live grants; 0 replay/live disagreements; 0 `RoleAdminChanged`; the timelock holds `DEFAULT_ADMIN` / `UPGRADER` everywhere; no EOA holds `CREDIT` / `MINTER` / `CONTROLLER` / `LOSS_BURNER` / `FEE_ACCOUNTING`; the deployer holds **0 of 306** probed pairs.
- **Parameters — zero drift** against frozen `Config.sol`, all five classes configured and active.
- **Storage — 16/16 ERC-7201 slots** correctly derived.
- **Static analysis — 194 Slither results reduce to the project's own 160-entry triaged baseline**; the two High families are its documented false-positives.
- **Access control — no gap.** 291 state-changing functions enumerated; every modifier-less one carries an explicit `msg.sender` check.

// 7. Reproduction

The fork harness binds to the live mainnet addresses (`Live.sol`) and drives the deployed contracts. Every finding is a passing test.

```
git checkout mainnet-v1-production-freeze-2026-08-16
# harness lives outside the tree; point forge at it:
export FOUNDRY_TEST=<corrovera-fork-dir> FOUNDRY_SRC=src MAINNET_RPC_URL=<archive endpoint>
cd contracts && forge build
forge test --match-path '<corrovera-fork-dir>/*' -vv
```

TEST	FINDING	RESULT
`Smoke.t.sol`	harness binds to live deployment; mint→deposit→originate→fund	3 pass
`C1_ProbeCliff.t.sol`	DV-01 healthy source misread → mark dropped to par	pass
`R2_IncidentLandmine.t.sol`	DV-02 namespace burn strands the arm	pass
`F03_EmptyCascade.t.sol`	DV-03 empty-cascade funding → senior loss	pass
`R1_ArmWedge.t.sol`	DV-05 arm freezes the queue	pass
`F02_GovFinality.t.sol`	§4.2 queued governance unstoppable	pass

TEST	FINDING	RESULT
`F01_CuratorNav.t.sol`	§3.9 F-01 correction	pass

On-chain verification (bytecode, roles, parameters, storage, distribution) used batched `eth_getCode` / `eth_getStorageAt` / `eth_getLogs` / `eth_call` at block 25,768,817 — scripts as listed in the deployment audit §12.

// 8. Recommendations, in priority order

1. **DV-01 / DV-06 together:** bound the conservative-mark ledger walk (cap live-event count or restore an O(1) aggregate; delete the vacuous reverse pass), and make the recovery probe budget scale with `gasleft()` so "unreadable" cannot mean "heavy." This is the one finding that silently under-marks the senior book.
2. **DV-03:** fund both cascade layers before the first origination — a hard §8 gate — and record it against register F-01.
3. **DV-02:** remove or namespace-isolate the legacy reserve-incident pair; add the documented deficit guard; make `closeReserveLossIncident` arm-aware.
4. **§4.1 / §4.2:** publish `SAFE-CD-01` and ADR-0036's queued-governance finality to the register; reword the deployment record's `CANCELLER` sentence to state the canceller is unreachable, not safely held.
5. **Operational keys:** fund the keeper and all four Safes; add a third attester; exercise the unused Safe key and both attester keys once before value moves.
6. **DV-04 / DV-07 / DV-08:** apply the points-hook floor + telemetry to the three curator hooks; size queue yield-recognition on outflow before vesting is enabled; correct the RC-01 comment. Add the `setCuratorModule` code-probe for future redeploys.
7. **DV-05:** gate `cancelAndDisable` on custody health rather than global solvency.
8. List the three unmanifested contracts; re-point the archived-manifest test fixture.

Corrovera Security · *Independent minds. Corroborated truth.* A report is evidence for human review, not proof that software is secure. Findings were reproduced against live mainnet bytecode on a fork; that is strong evidence of reachability, not a guarantee of exhaustiveness.